ChainPIM: A ReRAM-Based Processing-in-Memory Accelerator for HGNNs via Chain Structure

Wenjing Xiao, Jianyu Wang, Dan Chen, Chenglong Shi, Xin Ling, Min Chen, *Fellow, IEEE*, and Thomas Wu

***Abstract*—Heterogeneous graph neural networks (HGNNs) have recently demonstrated significant advantages of capturing powerful structural and semantic information in heterogeneous graphs. Different from homogeneous graph neural networks directly aggregating information based on neighbors, HGNNs aggregate information based on complex metapaths. ReRAM-based processing-in-memory (PIM) architecture can reduce data movement and compute matrix-vector multiplication (MVM) in analog. It can be well used to accelerate HGNNs. However, the complex metapath-based aggregation of HGNNs makes it challenging to efficiently utilize the parallelism of ReRAM and vertices data reuse. To this end, we propose ChainPIM, the first ReRAM-based processing-in-memory accelerator for HGNNs featuring high-computing parallelism and vertices data reuse. Specifically, we introduce R-chain, which is based on a chain structure to build related metapath instances together. We can efficiently reuse vertices through R-chain and process different R-chains in parallel. Then, we further design an efficient storage format for storing R-chains, which reduces a lot of repeated vertices storage. Finally, a specialized ReRAM-based architecture is developed to pipeline different types of aggregations in HGNNs, fully exploiting the huge potential of multilevel parallelism in HGNNs. Our experiments show that ChainPIM achieves an average memory space reduction of 47.86% and performance improvement by 128.29× compared to NVIDIA Tesla V100 GPU.**

***Index Terms*—Heterogeneous graph neural networks, processing-in-memory, resistive random access memory.**

Received 25 June 2024; revised 19 October 2024; accepted 3 January 2025. Date of publication 13 January 2025; date of current version 20 June 2025. This work was supported in part by the State Key Laboratory of Featured Metal Materials and Life-Cycle Safety for Composite Structures under Grant AA23073019; in part by the Guangxi Key Research and Development Plan Project under Grant AB23049006; in part by the Zhejiang Provincial Natural Science Foundation of China under Grant No.LY24F020014; and in part by the National Research Foundation, Singapore, under its Competitive Research Programme Award NRF-CRP23-2019-0003. This article was recommended by Associate Editor A. Gamatie. *(Corresponding author: Dan Chen.)*

Wenjing Xiao is with the School of Computer, Electronics and Information and the Guangxi Key Laboratory of Multimedia Communications and Network Technology, Guangxi University, Nanning 530004, China (e-mail: wenjingx@gxu.edu.cn).

Jianyu Wang, Chenglong Shi, and Xin Ling are with the School of Computer, Electronics and Information, Guangxi University, Nanning 530004, China (e-mail: jianyuw@st.gxu.edu.cn; chenglongs@st.gxu.edu.cn; lingxin@st.gxu.edu.cn).

Dan Chen is with the School of Computing, National University of Singapore, Singapore 119077 (e-mail: danchen@nus.edu.sg.

Min Chen is with the School of Computer Science and Engineering, South China University of Technology, Guangzhou 510006, China, and also with Pazhou Laboratory, Guangzhou 510330, China (e-mail: minchen@ieee.org).

Thomas Wu is with the School of Electrical Engineering, Guangxi University, Nanning 530004, China (e-mail: xwu@gxu.edu.cn).

Digital Object Identifier 10.1109/TCAD.2025.3528906

I. Introduction

GRAPH is a data structure that expresses complex relationships between objects. Graph neural networks (GNNs) map complex relationships between objects into a high-dimensional feature space to fully learn their intrinsic dependencies [1], [2], [3], [4] and achieve significant improvements in various applications, e.g., social network prediction [5], [6] and recommendation system [7], [8]. Due to the large number of irregular dependent accesses and the potential for high parallelism in GNNs, numerous studies have been proposed to accelerate them from the aspects of model optimization, software-level optimization, and hardware acceleration. In particular, recent works [9], [10], [11], [12], [13] demonstrate that resistive random access memory (ReRAM) is highly efficient for accelerating matrix-vector multiplication (MVM) and enabling in-memory computation, which reduces data movement by performing computations where the data is stored. This makes ReRAM an excellent platform for accelerating GNNs.

Existing works [3], [14], [15], [16] on accelerating GNNs using ReRAM focus on the homogeneous graph, which consist of only one type of vertices and edges. They follow the classic message passing framework [17], with each vertex aggregating information from its neighbors that directly connect to it based on graph structure. The homogeneous graphs do not have complex aggregating operations due to the limited types of vertices and edges. However, in real-world scenarios, heterogeneous graphs are more common, consisting of multiple types of vertices and edges with rich semantic information. For example, Fig. 1(a) shows the classic citation network of heterogeneous graphs, containing several types of vertices, such as *Paper (P)*, *Author (A)*, and *Conference (C)*. And as shown in Fig. 1(b), there are different relationships between them: *Author* $\overset{\text{write}}{\rightarrow}$ *Paper*, *Paper* $\overset{\text{cite}}{\rightarrow}$ *Paper*, and *Paper* $\overset{\text{presented at}}{\rightarrow}$ *Conference*. Furthermore, these relations can be composed to form high-level semantic relations, represented as metapath. For instance, *P-C-P* and *P-A-P* in Fig. 1(c) are two metapaths with different semantic information. The former indicates that papers were published in the same conference, and the latter indicates papers published by one author. Given a metapath M of a heterogeneous graph, a metapath instance m is a vertex sequence matched on the heterogeneous graph according to the defined metapath M [18]. For example, P1-A4-P1 is an instance of *P-A-P*, as shown in Fig. 1(c). Metapath-guided heterogeneous graph neural networks (HGNNs) utilize metapaths to aggregate

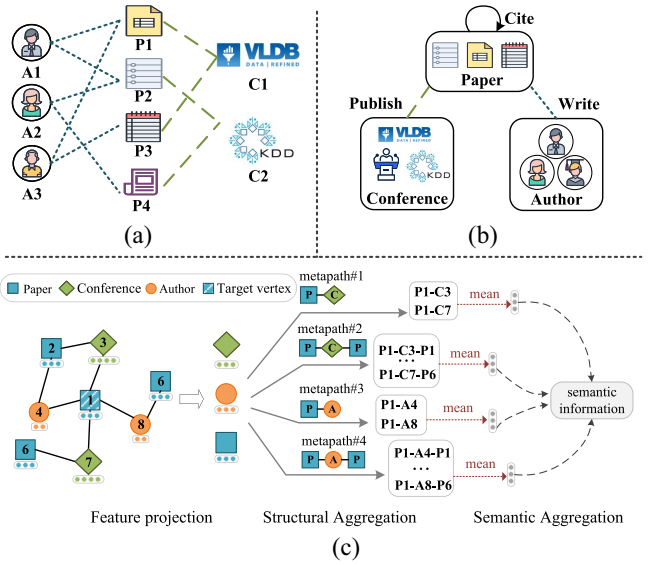

Fig. 1. Illustrative example of heterogeneous graphs. (a) Academic heterogeneous graph with vertex types author (A), paper (P), and conference (C). (b) Relationships between different types of vertices. (c) General processing flow of metapath-based HGNNs.

vertex information. They first aggregate vertex features within each metapath's scope to generate corresponding semantic vectors that capture different types of relational information. Then, these semantic vectors are fused to obtain the final embedding feature. This learned embedding feature incorporates multirelational information across the entire graph and is used for the final prediction task by the HGNN model. Unlike homogeneous graphs that directly aggregate information based on neighbors, heterogeneous networks perform more intricate aggregations based on metapaths.

There are only few works focusing on accelerating HGNNs. MetaNMP [19] investigates the problems of redundant computation and high-memory footprint in HGNNs, and leverages Near-Memory Processing to solve them. However, due to its need for intricate control logic to achieve reusing complex metapaths, this work suffers from serious scalability limitations. Benefiting from the efficient storage and in-situ computing capabilities of ReRAM, it significantly reduces the complexity of control logic introduced by processing complex data like heterogeneous graph data. But research on ReRAM-based schemes has mostly focused on homogeneous graphs, which are difficult to directly use to accelerate more complex HGNNs efficiently. They do not take the following three key aspects of HGNNs into account for a ReRAM-based accelerator.

1) *Instance-Level Parallelism:* In addition to vertex and feature parallelism in traditional GNNs, metapath instances can be processed and executed in parallel. Hence, ReRAM can perform different metapath instances computation simultaneously.
2) *Metapath-Level Parallelism:* Different metapaths are independent of each other. Such independence means that not only instances within the same metapath can be performed in parallel, but also instances of different metapaths can be executed simultaneously.
3) *Vertex Data Reuse:* Vertex data can be reused because there exist multiple same vertices across different metapath instances. Reusable vertex data can be shared directly in the crossbar for different metapath instances.

Considering above HGNN features is important for improving HGNN execution efficiency on ReRAM-based processing-in-memory architecture.

To this end, we propose the first ReRAM-based HGNNs accelerator, named ChainPIM, a novel integrated architecture to efficiently process HGNNs with high parallelism and significant reuse of vertex data. ChainPIM features three novel designs. We first construct chain structures from related metapaths. Then we generate relationships among instances involved in metapaths on the chain structure, named as R-chain. Different R-chains can be processed in parallel, and the metapath instance relations expressed by R-chain can help us efficiently reuse the vertex data. Second, to achieve efficient storage of R-chains, ChainPIM further develops an effective yet novel storage format for R-chain, which utilizes the instance relations expressed by R-chain to eliminate redundant vertex storage. Third, ChainPIM introduces a ReRAM-based hardware architecture with a multilevel aggregation process, fully exploiting the potential of computational parallelism in HGNNs by pipelining aggregation stages. Our main contributions of this article are as follows.

1) We introduce R-chain to build related metapath instances together based on chain structure. It is not only to explore parallelism but also to exploit vertex data reuse.
2) We develop an efficient storage format for R-chain, which reduces a lot of repeated vertices storage.
3) We present the first ReRAM-based PIM architecture for HGNNs, which pipelines different types of aggregations in HGNNs to exploit the multilevel parallelism.
4) We evaluate our accelerator with representative HGNNs models on extensive graph datasets. Experimental results on ReRAM crossbar sized of 128 $\times$ 128 show that ChainPIM achieves an average memory space reduction of 47.86% and performance improvement by 128.29$\times$ compared to NVIDIA Tesla V100 GPU.

II. Background and Motivation

A. Heterogeneous Graph Neural Networks

Traditional homogeneous GNNs only handle input graphs with one type of vertex and edge. However, real-world graphs are usually heterogeneous, containing multiple types of vertices and edges. To capture the complex structural and semantic features in these graphs, scholars have proposed heterogeneous GNNs [20], [21], [22] with strong capability to model and learn high-dimensional relations. A heterogeneous graph consists of vertices, edges, and several mapping functions, denoted as $\mathcal{G} = \{\mathcal{V}, \mathcal{E}, \mathcal{T}^v, \mathcal{T}^e\}$. Among them, $\mathcal{V}$ represents the set of vertices associated with a vertex type mapping function $\psi : \mathcal{V} \rightarrow \mathcal{T}^v$, and $\mathcal{E}$ represents the set of edges associated with an edge type mapping function φ: $\mathcal{E} \rightarrow \mathcal{T}^e$. In metapath-based HGNNs, the metapath P is defined as a fundamental path in the form of $V1 \xrightarrow{R_1} V2 \xrightarrow{R_3} \cdots \xrightarrow{R_l} V_{l+1}$, abbreviated as $V_1V_2, \ldots, V_{l+1}$, which describes a composite

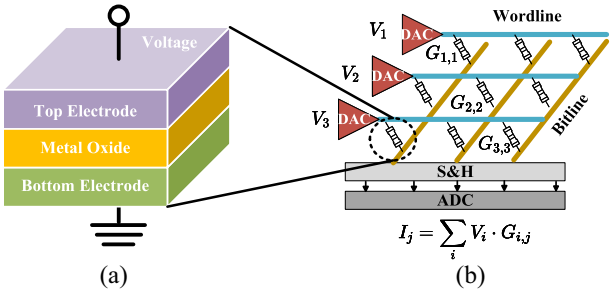

Fig. 2. Structure of (a) ReRAM cell and (b) ReRAM crossbar.

relation $R = R_1 \circ R_2 \circ \cdots \circ R_l$ between the vertex type V_1 and V_{l+1}, where $\circ$ represents the composition operator.

Generally, heterogeneous graph neural networks, as a type of GNN model, are used to learn representations of heterogeneous graphs. Let $\mathcal{N}_v^P$ denote metapath-based neighbors of vertex v, to express the set of vertices connecting with vertex v through the metapath instance P. To this end, a layer of HGNN computation on a vertex v can be formulated as

$$\mathbf{h}_v^P = \text{Transfor}_{\psi(v)}\left(\text{Reduce}(\{\text{Agg}_p(\mathcal{N}_v^r)\})\right) \tag{1}$$

where $r \in \mathcal{N}_v^P$ and $P \in \mathcal{P}_v$. The $\mathcal{P}_v$ denotes the set of metapaths on vertex v and $\text{Agg}_p(\cdot)$ is a neighborhood aggregation function specific to the metapath type. Reduce$(\cdot)$ is a function to combine (e.g., mean/sum) aggregation results along the metapath instance, after aggregating information intra and intermetapath. $\text{Transfor}_{\psi(v)}(\cdot)$ is a transformation function specific to vertex-type of v parameterized by θ_ψ. The above procedure in (1) is a general process of HGNNs.

In summary, the HGNN framework is structured into three principal stages: 1) feature projection; 2) structural aggregation; and 3) semantic aggregation, illustrated in Fig. 1(c). Initially, to standardize the feature dimensions for both vertices and edges, the feature projection stage adjusts all features to a uniform dimensional space. Subsequently, the process of structural aggregation is deployed to consolidate structural data, which includes two distinct substages: 1) the aggregation of features within a single instance (intrainstance aggregation) and 2) the aggregation of features across different instances within a single metapath (interinstance aggregation). Lastly, in the semantic aggregation phase, HGNNs integrate features from multiple metapaths to construct a comprehensive semantic context.

B. ReRAM-Based Processing-in-Memory

Processing-in-memory (PIM) is a computational architecture that enables memory cells to process data, thereby overcoming the data movement bottleneck between the processor and memory [23]. In particular, as a typical PIM architecture, Resistive Random Access Memory (ReRAM) [24] is a type of nonvolatile memory that operates by switching the resistance of a ReRAM cell. Fig. 2(a) depicts a ReRAM cell with a metal-insulator-metal structure [15], where high-resistance state and low-resistance state are used to represent logic "1" and "0," respectively. The ReRAM crossbar architecture provides powerful parallel execution of in-situ MVM operations [25], [26]. As shown in Fig. 2(b), the matrix data $M_{i,j}$ are programmed to conductance $G_{i,j}$ in advance, where i, j are the indices of row and column in M. Then, each dimension of input vector N_i can be converted to the analog voltage V_i by applying digital-to-analog converters (DACs), where i is the dimension index of the input vector. The cell (i, j) passes the current $V_i \cdot G_{i,j}$ to the bitline and the cumulative currents flowing to the end of the same bitline can be regarded as dot product operation $I_j = \sum_i V_i \cdot G_{i,j}$, based on Kirchoff's Current Law. This result can be converted to a digital value and is equivalent to the dot product operation $O_j = \sum_i N_i \cdot M_{i,j}$. By collecting every dimension of O on the corresponding bitline, the MVM operation $O = NM$ can be efficiently performed using a crossbar array [27].

Based on the above features, ReRAM-based PIM is naturally a suitable solution for HGNNs. There are three reasons for this. First, ReRAM-based processing can execute MVM operations with varying sizes in an efficient manner, adapting to the process of HGNNs that involves multilevel, varying sizes and numerous MVMs for aggregating features. Second, the powerful parallelism of the ReRAM architecture can fully exploit intrainstance, interinstance, and intermetapath parallelism in HGNNs. Third, by leveraging PIM, ReRAM-based processing can overcome the memory bottleneck of HGNNs, as it enables data processing directly within the ReRAM, significantly reducing the need for data movement between the compute unit and memory.

C. Motivation

As described in Section II-A, HGNNs have more complex processing stages compared to traditional homogeneous graph neural networks, consisting of distinct feature projection, structural aggregation, semantic aggregation, etc. Therefore, we conduct a series of experiments to analyze the performance characterization of HGNNs under existing ReRAM-based architecture designed for traditional GNNs, which reveals the inefficiency of HGNNs execution on existing ReRAM-based architecture and highlights the need to design new ReRAM-based architectures for HGNNs.

Observation 1: Only vertex and feature parallelism is utilized. Traditional GNNs have vertex-level and feature-level parallelism, hence existing ReRAM-based GNNs accelerators focus on the vertex and feature-level parallelism [3]. However, since aggregating information in HGNNs relies on metapaths, in addition to vertex-level and feature-level parallelism, HGNNs have metapath-level and instance-level parallelism. Fig. 3(a) provides a breakdown of the inference time for HGNNs, revealing that structural and semantic aggregations account for more than 90% of the total time. This indicates that instance-level and metapath-level aggregations are the primary time-consuming processes in HGNNs. Thus, improving the execution efficiency during aggregation stages is crucial for accelerating HGNN computation. Fig. 4(c) further shows the importance of exploiting instance-level and metapath-level parallelism. It can be seen that if we ignore these two levels of parallelism with existing ReRAM-based architectures to accelerate HGNNs directly, it will lead to inefficient and

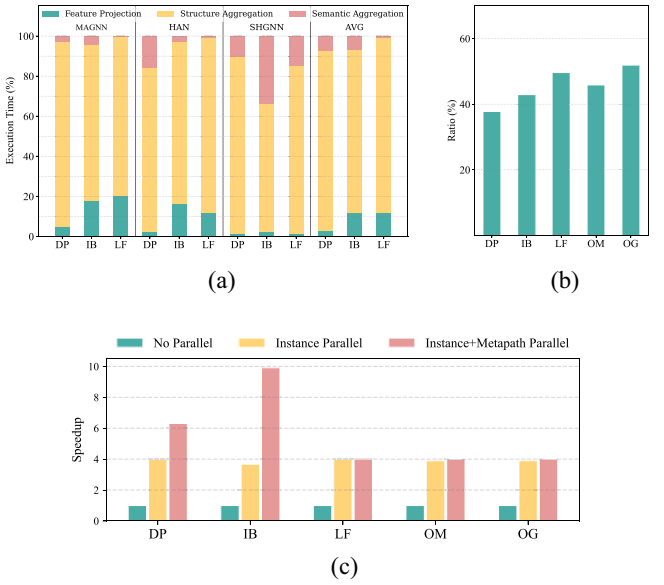

Fig. 3. (a) Breakdown of execution time and (b) ratio of repeated vertex load in HGNN inference. (c) Experimental results of adding instance-level and metapath-level parallelism.

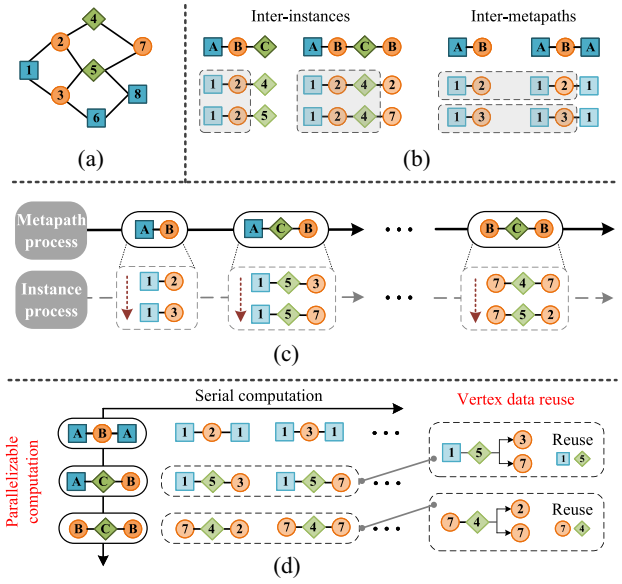

Fig. 4. Illustration of the main idea of ChainPIM. (a) Initial graph. (b) Repeated vertex load. (c) Serial computation of metapath instances. (d) Insight of ChainPIM.

suboptimal performance. To further explain, we take ①-②-①, ①-⑤-③, and ⑦-④-② three metapath instances as an example. There are no dependencies among them. If their parallelism is not exploited, executing them sequentially would take much more overhead than executing them in parallel. Fig. 3(c) shows the performance of adding instance-level and metapath-level parallelism on five different datasets. Experimental results illustrate the significant impact of both instance-level and metapath-level parallelism on the acceleration of HGNNs, demonstrating the potential of leveraging these parallelisms to achieve optimal performance.

Observation 2: Partial extensibility in metapath instances causes severe redundant accesses of vertices. We observe that there are partial extensions between metapaths in HGNNs. Specifically, a metapath is generated by adding new types of vertices and heterogeneous edges over another metapath to describe more complex semantic relations. Thus, the short metapath instance are identical to the part of long metapath instance. In this case, it results in the same vertices being loaded into the ReRAM repeatedly. As shown in Fig. 4(b), ①-②,①-③ and ①-②-④ are loaded more than once, causing a waste of hardware resource. Further, we analyze the repeated loading across five different datasets on MAGNN (a typical HGNN) [18] as depicted in Fig. 3(b). The experimental results reveal that there is a high ratio of repeated loading on all datasets. This significantly hampers performance, making it crucial to eliminate these inefficiencies for ReRAM.

Based on the above observations, ChainPIM aims to fully exploit the potential of parallelism in HGNNs and significantly reuse the vertex data during the execution of HGNNs as shown in Fig. 4(d). However, it can be seen that achieving an efficient ReRAM-based PIM architecture for HGNNs still presents many challenges. On the one hand, it is difficult to efficiently capture the relationships of metapath instances. Their relationships is critical to exploit parallelism of unrelated instances. On the other hand, how to utilize reusable vertex data of HGNNs in ReRAM architecture to reduce redundant data load is another challenge. To this end, we propose the first ReRAM-based HGNN accelerator, named ChainPIM, a novel PIM architecture to efficiently accelerate HGNNs with high parallelism and significant vertex data reuse.

III. Overview of Our Solution

In this section, we first introduce the construction of R-chain to support exploiting parallelism and vertices reuse in ReRAM-based architecture. Then, we design an efficient storage format for R-chain.

A. Construction of R-Chain

To exploit parallelism and vertices reuse in HGNNs, an intuition is that metapath instances involving repetitive vertices can be distributed together to reuse vertex data while unrelated metapath instances can be processed in parallel. To achieve this, ChainPIM organizes metapaths into distinct groups and introduces a chain structure to capture relationships of metapaths in each group. Metapath instances in different groups can be processed in parallel, and instances in the same group use the chain structure to exploit vertices reuse. In this work, we propose a novel method for creating groups.

We draw inspiration from Radix sort [28], which is a sorting algorithm that distributes elements into buckets based on their individual characters at key positions. Following a similar method, we can asymptotically assign metapaths to different groups based on their vertex type at the key position. After obtaining groups, we construct chain structures of metapaths in each group to exploit vertices reuse.

Specifically, given the set of metapaths, we first choose the starting position of the metapath as the key position, and metapaths that have the same vertex type at the key position will be assigned to the same group. Then, we move the key

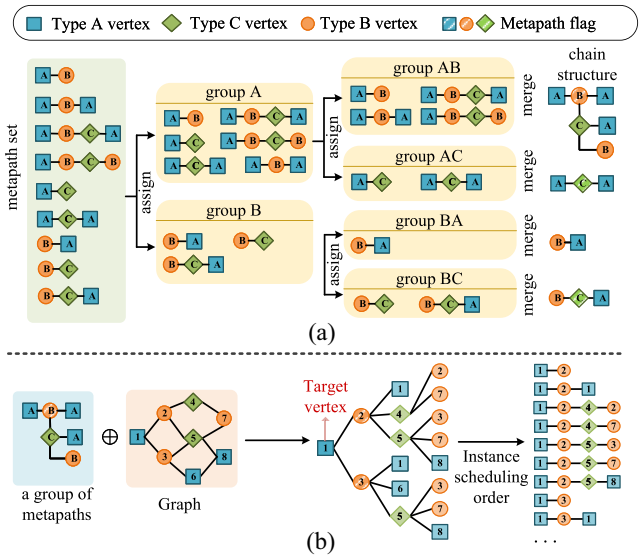

Fig. 5. Construction of R-chain. (a) Organizing metapaths into distinct group. (b) Generating relationships among metapath instances.

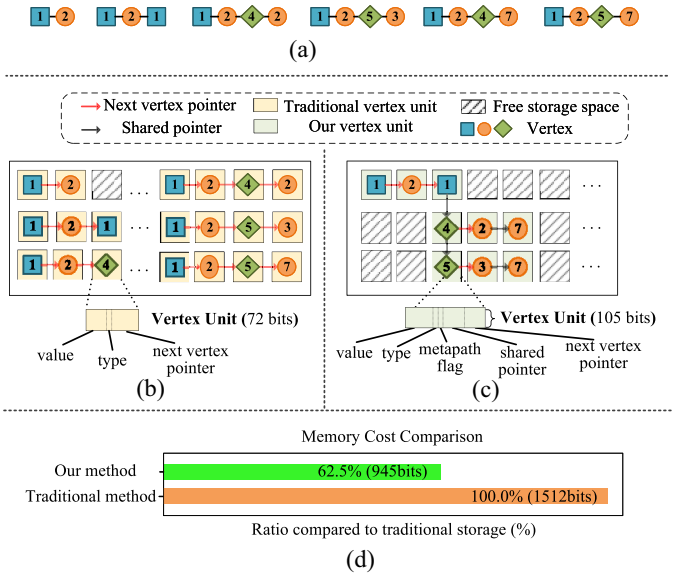

Fig. 6. Storage of instance. (a) Storaged metapath instances. (b) Traditional storage of instances. (c) R-Chain based storage of instances. (d) Comparison between traditional storage and R-Chain based storage.

position forward to the next position, and metapaths in the same group will be further assigned to different groups based on the vertex type at the key position. Afterward, the key position moves forward, and the groups will be divided again. This process will be repeated several times, and we will get the final groups. The more groups are divided, the more metapaths can be executed in parallel. The number of repetitions of the above process is defined as n. Take n equals to 2 as an example, as shown in Fig. 5(a), given the original set of metapaths, we let starting position as the key position, and the metapath will be assigned to group `A` or `B` according to their vertex type at the key position. Then, the key position moves forward to the next position. Group `A`, `B` are further divided into group `AB`, `AC` and `BA`, `BC` separately, and the metapaths are further assigned to corresponding groups based on their vertex type at the key position. Finally, we merge the same vertex type that appears in the same position of different metapaths in each group into one and obtain a chain structure. The repetition factor n represents a tradeoff between parallelism and vertex reuse. A larger n provides more parallelism while a smaller n more effectively promotes vertex reuse.

Based on the chain structure in each group, we can generate relationships among metapath instances, called R-chain, to reuse vertex data for efficient execution of HGNNs. Fig. 5(b) shows an example of how to obtain a R-chain based on chain structure. Given a specific graph structure, we choose a vertex ① of A type as the target vertex. According to the chain structure, the successor of A is B, so we first need to find the B type neighbors of ①, i.e., ② and ③, and attach them to the ①. Subsequently, since the successors of B are A and C, we then find A and C neighbors of ② and ③, i.e., ①,④,⑤ and ①,⑥,⑤. They are attached to ② and ③, respectively. This process is repeated until the chain structure is perfectly matched. Finally, we obtain the corresponding R-chain.

In a word, chain structure provides a simple and effective guidance to generate R-chain in each group. Traversing and executing along different R-chains at runtime enables the parallelism of HGNNs in a natural way. Specifically, after constructing R-chains, each becomes an autonomous task that can be processed efficiently in parallel. This approach facilitates parallel processing across different instances and metapaths, thereby enhancing overall throughput.

B. Instances Storage Over R-Chain

Above constructed R-chains are to be prestored in the off-chip memory to support parallel execution of HGNNs. In subsection, to achieve efficient storage of R-chains, we design an efficient storage format for them.

For the traditional method of storing metapath instances, as shown in Fig. 6(b), each vertex consists of three parts of vertex value, vertex type, and next vertex pointer, which are int (4B), char (1B), and int (4B), respectively. Vertices in a metapath are connected by the next vertex pointer, and if the next vertex pointer of a certain vertex is null, it indicates that this vertex is a tail vertex of this instance. To store R-chains, a straightforward approach is to add connections between instances in the traditional storage method. However, this approach introduces redundant vertex storage and doesn't fully exploit the storage benefits that R-chain itself offers. Therefore, we propose a more efficient storage format for R-chain, as shown in Fig. 6(c). We add a metapath flag (1 bit) and a shared pointer (4B) on each vertex storage. The metapath flag marks the end of a metapath instance and is used to identify short metapath instance. For example, vertex ② in instance ①-②-① is not the end of the chain, but the metapath flag of vertex ② will be set to "1" to indicates there is a short metapath ①-②. The shared pointer is used to connect the vertices that share the same prefix structure, for example, the shared pointer of ① points to ④ because ①-②-① and ①-②-④ share the same prefix ①-②. The shared pointer of ③ points to

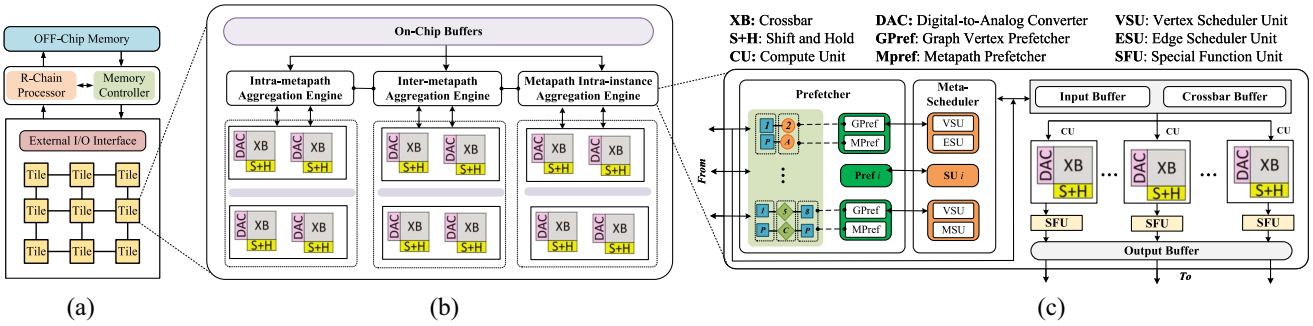

Fig. 7. ChainPIM architecture. (a) Overview of ChainPIM. (b) Structure of ChainPIM Tile. (c) Metapath Intra-instance Aggregation Engine.

⑦ because ①-②-⑤-③ and ①-②-⑤-⑦ share the same prefix ①-②-⑤. Although the additional metapath flag and shared pointer increase the memory consumption from 72 bits to 105 bits for each vertex, our method still significantly reduces the overall memory overhead compared to the traditional storage. This is because traditional storage of instances stores all vertices of each instance, involving a lot of repeated vertex storage. We eliminate repeated vertex storage in our approach. As shown in Fig. 6(d), our approach saves more than 30% memory compared to the traditional storage of instances under the given example in Fig. 6(a). Overall, our proposed storage approach can efficiently store the R-chain for each group and also reduce storage overheads.

IV. ChainPIM Architecture

In this section, we first introduce the overall architecture of ChainPIM and its workflow. Then, we present a scheduling strategy to make ChainPIM more efficient by improving vertex reuse and eliminating redundancy.

A. Overview of ChainPIM Architecture

Fig. 7(a) presents the overall architecture of ChainPIM, which is organized in a hierarchical manner. A ChainPIM chip comprises a number of ReRAM tiles interconnected through a mesh-based internal communication network. Each tile includes multiple compute units (CUs). Additionally, ChainPIM can also connect to large off-chip memory, which is ideal for handling large graphs that are sequentially streamed into the tiles via a memory controller. Furthermore, a R-chain processor is laid out between off-chip memory and ChainPIM chip, enabling the fast construction of R-chain without the need for a host processor. We describe each component of ChainPIM below.

Off-Chip Memory: Due to inherent material and architectural limitations, the memory capacity of computational ReRAM crossbars is typically limited and unable to accommodate large graph datasets. Therefore, off-chip memory is employed in ChainPIM and primarily used to store input and output data of HGNNs. The input data includes metapaths, graph structure information, R-chains, and vertex features. Metapaths and graph structure information are used by R-chain processor to generate R-chains. Then meta-scheduler uses R-chains to manage task execution (Section IV-C) during the execution of HGNNs. Additionally, vertex features are stored in a dense matrix format to enable their fast load on ReRAM crossbars without the format conversion overhead.

ChainPIM Tile: As shown in Fig. 7(b), a ReRAM tile consists of a set of on-chip buffers and three types of aggregation engines. The on-chip buffer is composed of three distinct components: 1) an input buffer, which temporarily holds weight data that feeds into the crossbar as an input; 2) a crossbar buffer designed to cache vertex features prior to their mapping into the crossbars; and 3) an output buffer, which temporarily stores outputs produced by the Special Function Units (SFU). Three types of aggregation engines are metapath intrainstance aggregation engine (MIAE), intrametapath aggregation engine (IAAE), and intermetapath aggregation engine (IEAE). These engines are used for the three levels of aggregations, respectively. Fig. 7(c) shows the microstructure of MIAE. This engine comprises a prefetcher, a meta-scheduler, on-chip buffers, multiple CUs and their corresponding SFUs. The meta-scheduler manages the scheduling order of metapath instances and sends them to the prefetcher. This scheduler adopts our scheduling strategy (Section IV-C) to generate scheduling orders, achieving high-data reuse and low-vertex loading overhead. The prefetcher receives the optimized scheduling order from the meta-scheduler and then prefetches the corresponding metapath and vertex features in order. Since the prefetching process is covered by the HGNN execution stage, its overhead is negligible.

R-Chain Processor: R-chain processor works independently of the ReRAM for constructing R-chains to be stored in the off-chip memory. Instead of constructing all the R-chains before aggregation, ChainPIM adopts a gradual generation manner. This means pipelining the generation of R-chains in the R-chain processor and the aggregation in ReRAM. R-chain processor generates and stores enough R-chains for computations required for ReRAM. Memory allocated for storing R-chains is released after ReRAM completes corresponding computations, which significantly reduces the memory footprint.

CUs: A CU consists of core computational logics and peripheral circuits, such as digital-to-analog converter (DAC), to support matrix-vector multiplication (MVM). Since the main operations of the three-level aggregations in HGNNs are MVMs, our ReRAM crossbar-based CU can aggregate features of multiple vertices or instances simultaneously, achieving efficient execution of HGNNs.

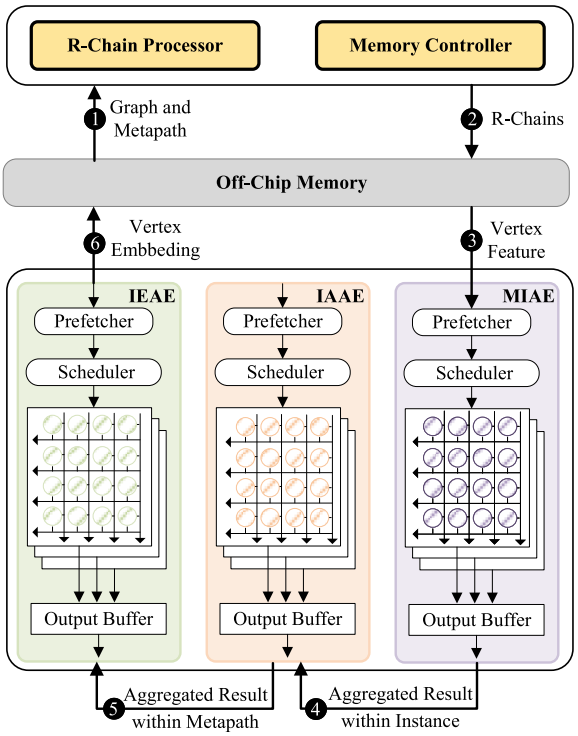

Fig. 8. Workflow of ChainPIM.

Memory Controller: The memory controller coordinates the data transfers between the off-chip memory and the ChainPIM chip. It ensures that input data is prefetched by the prefetcher to on-chip buffers, thus hiding the memory access latency. To boost efficiency, the final output results are batch-written back to the off-chip memory. Furthermore, intermediate results generated by the three aggregation engines seamlessly propagate across various levels of aggregation engines by leveraging the on-chip network.

B. Workflow of ChainPIM

In essence, ChainPIM is a task-pipelined accelerator, where the workflow of ChainPIM is shown in Fig. 8. Specifically, given a vertex, based on graph data and metapath ❶, the R-chain processor generates R-chains and stores them in the off-chip memory ❷. These generated R-chains are used to guide the aggregation process, enabling both interinstance and intermetapath parallelism. After R-chains are generated, the input data transfers from off-chip memory to the MIAE ❸. The weight parameters are loaded into input buffer as an input vector, while the vertex features are loaded into crossbar buffer as a matrix. Note that the loaded vertex features are predetermined when the scheduler establishes the task order. This design allows ChainPIM to utilize a prefetching mechanism, effectively hiding the off-chip memory access latency. Then, multiple CUs in MIAE perform matrix-vector multiplication (MVM) operations to aggregate the metapath-based neighbors of the given vertex and send the partial results to SFU. The SFU consists of a shift-and-add unit (S&A) and a scalar arithmetic and logic unit (sALU), which are utilized to further process the partial results and send them to output buffer. Once the aggregation of an instance is completed, the intermediate results will be written to the IAAE to calculate the aggregation result of the corresponding metapath ❹. Subsequently, the aggregation results of all metapaths associated with the given vertex are sent to the IEAE to compute the embedding of the given vertex ❺. Finally, the vertex embedding will be written back to the off-chip memory ❻.

While the three engines operate in a pipelined manner, the potential of imbalanced workload among them can degrade the utilization of ReRAM crossbars, thereby reducing the system throughput. To allocate proper number of crossbars to MIAE, IAAE and IEAE, we adopt a sample-based method to estimate the workload of each engine, and the workload ratio is estimated by the following equation:

$$W_1{:}W_2{:}W_3 = \sum_{i=1}^{M} \lceil L_i/C_r \rceil \cdot (w \cdot I_i + r) : \sum_{i=1}^{M} \lceil I_i/C_r \rceil \cdot (w \cdot C_r + r) : \lceil M/C_r \rceil \cdot (w \cdot C_r + r) \quad (2)$$

where C_r is the row size of crossbar, w and r represent the number of cycles required per write and read operation, respectively, L_i is the length of metapath i, and M and I_i are the number of metapaths and instances for metapath i, respectively. We sample workload ratios of 100 vertices for each metapath type and adopt their mean to determine the number of crossbars allocated to MIAE, IAAE and IEAE. This approach enables ChainPIM to adaptively coordinate varying hardware specifications and workload patterns, achieving balanced workload and high-hardware utilization at negligible cost.

C. Locality-Aware Instance Scheduling

In this subsection, we discuss how ChainPIM reuses vertices and then use a case study to further illustrate the vertices reuse in the ReRAM architecture.

Based on R-chain, we take the locality into account and adopt a locality-aware instance scheduling method. As shown in Fig. 5(b), we utilize a depth-first search (DFS) on R-chain to generate an instance scheduling order for target vertex ①. This approach traverses and records each path from the target vertex to the end vertices of metapath instances. This is reasonable because adjacent paths share more prefix vertices and exhibit greater locality. As illustrated by a simple example in Fig. 9, given the original graph and corresponding metapath instances, the metapath-ordered instance scheduling shown in Fig. 9(c) loads the intermediate vertices ②,③,④ and ⑤ multiple times, resulting in redundant loads (a total of 13 loads). Conversely, our locality-aware scheduling approach shown in Fig. 9(d) eliminates such redundancies, reducing the number of loads to just 7 by leveraging our generated R-chain. In this way, vertices data are fully reused.

Vertices Reuse in Crossbar: Fig. 10 shows how to reuse vertices data in the ReRAM architecture. When we aggregate metapath ①-③, the prefetcher (Section IV-A) in ChainPIM will load their features by row, and each column represents a

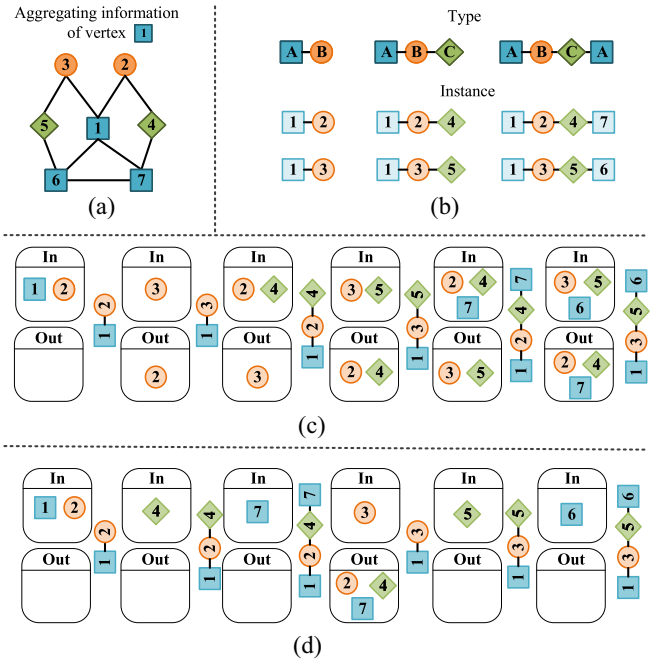

Fig. 9. Comparison of metapath-ordered instances scheduling and locality-aware instances scheduling. (a) Original graph. (b) Metapath types and instances. (c) Metapath-ordered instances scheduling (13 loads). (d) Locality-aware instances scheduling (7 loads).

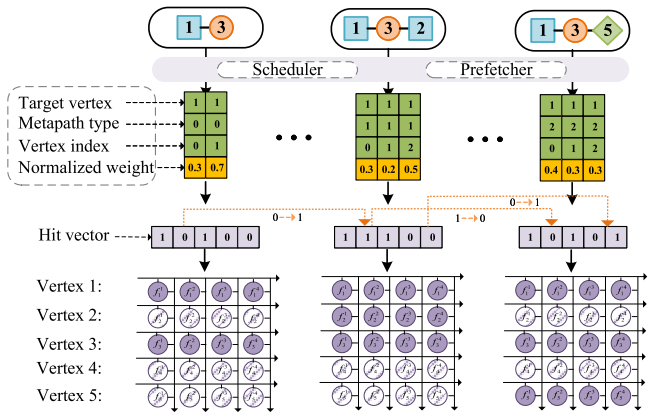

Fig. 10. Illustration of how crossbar-based processing reuses the vertex data.

dimension of the feature. Subsequently, the ReRAM crossbar utilizes the target vertex ① and metapath type A-B to search for the normalized weights (i.e., 0.3 and 0.7) and generates a hit vector (i.e., 1-0-1-0-0), which are then passed to the ChainPIM controller. Finally, the ChainPIM controller will activate the row in the crossbar according to the hit vector, feed the normalized weight as an input vector, and perform an MVM operation, efficiently aggregating ①-③ within a single cycle. When it comes to aggregating ①-③-②, crossbar-based processing does not store the intermediate result of ①-③ for reusing. It only reuses the vertex feature remaining in the crossbar (i.e., ① and ③) and simply maps the feature of ② into the crossbar. Subsequently, ChainPIM searches the corresponding normalized weights (i.e., 0.3, 0.2, and 0.5), changes the hit vector and performs an MVM operation for aggregating ①-③-②.

TABLE I
CHAINPIM CONFIGURATION

Component	Param.	Spec.	Power	Area (mm^2)
CU Properties (24 CUs per Tile)				
ADC	resolution	8 bits	64 mW	0.0384
	number	32		
DAC	resolution	1 bit	16 mW	0.00068
	number	32 × 128		
Crossbar Array	bits per cell	2 bits	9.6 mW	0.0008
	size	128 × 128		
	number	4 × 8		
S+H	number	32 × 128	40 μW	0.00016
S&A	number	32	1.6 mW	0.0008
IR	size	8 KB	3.04 mW	0.0062
OR	size	1 KB	0.48 mW	0.0019
Tile Properties (32 Tiles per Chip)				
CUs	number	24	2274.24 mW	1.175
CB	size	64 KB	17.66 mW	0.0086
IB	size	8 KB	1.91 mW	0.0054
OB	size	8 KB	1.91 mW	0.0054
SFU	Total	24	6.12 mW	0.0072
Scheduler & Prefetcher	-	-	58.07 mW	0.084
Chip Properties				
Tiles	number	32	75.52 W	41.14
Controller	-	-	580.41 mW	2.65
Chip Total	-	-	76.1 W	43.79

V. EVALUATION

A. Experimental Setup

Simulation: We develop a cycle-accurate simulation platform for ChainPIM, integrating comprehensive models for computation, timing, and energy consumption across all components. The simulation framework features an assembly of 32 ReRAM tiles, each hosting 24 compute units (CUs). Each CU includes 32 ReRAM crossbars. Each crossbar is sized of 128×128 and utilizes TaOx 0T1R-type ReRAM cells with read and write latencies of 29.31 and 50.88 ns, respectively [29]. Additionally, to reduce sensing pressure and potential error probabilities introduced by the resistance variability of ReRAM elements, we choose a conservative 2-bit multilevel cell architecture [25]. We configure off-chip memory as high-bandwidth memory (HBM) with a capacity of 128 GB and a bandwidth of 900 GB/s. Timing simulations for memory accesses are conducted using Ramulator [30], where the per-bit energy consumption adheres to specifications at 7 pJ [31]. The microarchitectural specifics within each tile are systematically modeled using CACTI 7 [32], which assesses the area, power, and access latency of all internal buffers. Additionally, the scheduler and prefetcher of ChainPIM, crafted in Verilog RTL, are synthesized utilizing the Synopsys toolchain within the TSMC 40-nm technology framework. The ADCs and DACs are configured at resolutions of 8 and 1 bit, respectively, and their area and energy evaluations are based on [25]. The fundamental configuration of ChainPIM is summarized in Table I, which includes the power and area specifications for each component.

Methodology: We compare ChainPIM with four state-of-the-art designs on typical platforms: 1) *Baseline:* Intel Xeon Gold 5117 CPU; 2) the NVIDIA Tesla V100 GPU; 3) REFLIP [3], a ReRAM-based accelerator designed for GCN; and 4) MetaNMP [19], a near-memory processing accelerator for HGNNs. Hardware specifications of CPU and GPU are shown in Table II. Additionally, the estimation of

TABLE II
CPU AND GPU SPECIFICATIONS

CPU Specifications	
CPU	Intel Xeon Gold 5117 Processor
CPU Cores	14-Cores, 28-threads, 2.00 GHz
Cache Per Core	L1: 64KB; L2: 1024KB; L3: 1.375MB
Main Memory	128GB DDR4

GPU Specifications	
GPU	Nvidia Tesla V100
GPU Cores	80 SMs, 64 CUDA cores Per SM, 1.38GHz
Cache	128KB L1 Cache (per SM), 6MB L2 Cache
Main Memory	16GB HBM2

TABLE III
DATASETS

	#Vertex	#Edge	Metapath
DBLP (DP) [36]	Author(A): 4057 Paper(P): 14328 Term(T): 7723 Venue(V): 20	A-P: 19645 P-T: 85810 P-V: 14328	APA APTPA APVPA
IMDB (IB) [19]	Movie(M): 4278 Director(D): 2081 Actor(A): 5257	M-D: 4278 M-A: 12828	MDM MAM DMD DMAMD AMA AMDMA
LastFM (LF)[19]	User(U): 1892 Artist(A): 17632 Tag(T): 1088	U-U: 25434 U-A: 64984 A-T: 23253	UAU UATAU AUA ATA
OGB-MAG (OM) [37]	Author(A): 1134649 Paper(P): 736389 Institution(I): 8740 Field(F): 59965	A-I: 1043998 A-P: 7145660 P-P: 5416271 P-F: 7505078	APA APFPA
OAG (OG) [38]	Author(A): 5985759 Paper(P): 5597605 Institution(I): 27433 Field(F): 119537 Venue(V): 16931	A-I: 7190480 A-P: 15571614 P-P: 5597606 P-F: 47462559 P-V: 31441552	APA APFPA

CPU energy consumption is derived from the Intel product specifications [33], while the energy usage of the GPU is ascertained using the NVIDIA system management interface (Nvidia-SMI) [34]. Note that we separate out the time of metapath instance generation in MetaNMP, leaving only the inference time for comparison.

Workloads: Table III shows details of the heterogeneous graph datasets used in our experiment, including DBLP (DP) [35], IMDB (IB) [18], LastFM (LF) [18], OGB-MAG (OM) [36], and OAG (OG) [37] datasets. Additionally, we consider three representative HGNNs.

1) *MAGNN* [18] aggregates vertices within the metapath instance for structural information and metapaths associated with the target vertex for the semantic information.
2) *HAN* [38] only aggregates metapath-based neighbors (i.e., the end vertex of the metapath instance) for structural information and performs semantic aggregation on the results of structural aggregation to obtain the final vertex embedding, employing vertex-level and semantic-level attentions.
3) *SHGNN* [39] aggregates vertices within the tree structure for structural information and performs semantic aggregation across different trees.

B. Performance

We first compare the performance of ChainPIM with different platforms. Then, we show benefits of the ChainPIM in terms of energy efficiency. Fig. 11 shows the performance of ChainPIM against CPU, GPU, REFLIP [3], and MetaNMP [19]. The details are as follows.

ChainPIM Versus CPU: ChainPIM outperforms CPU by 1320× on average, due to its highly parallel in-situ MVM operations in ReRAM crossbars and software-hardware co-design to efficiently capture the relationships of metapaths and aggregate metapath instances. Specifically, the benefits obtained for HGNNs are highly related to models. As we can see, ChainPIM has a higher-performance improvement in MAGNN than HAN and SHGNN. This is because MAGNN has more irregular and repeated vertex accesses. Therefore, MAGNN shows a 1401× performance improvement on average.

ChainPIM Versus GPU: GPU is the most commonly used platform to accelerate HGNNs. Compared with GPU, ChainPIM is 128.29× faster on average. Although the GPU features an extensive array of CUDA cores (5120) and high-bandwidth memory (HBM), the irregular memory accesses and data dependencies within HGNNs challenge the effective exploitation of its vast parallelism capabilities. In contrast, ChainPIM addresses the above issues effectively by fully leveraging locality in metapaths and potential of multilevel parallelism, thereby maximizing the benefits of in-situ MVM operations in ReRAM crossbars. In particular, the benefits for HGNNs are highly related to graph structure and metapaths. DBLP shows the highest-performance improvement of 164.61× on average, since its irregular memory accesses make GPU hard to accelerate and there is greater locality in its metapaths to be exploited by our locality-aware scheduling. These factors collectively contribute to the substantial performance gains observed. In addition, ChainPIM utilizes processing-in-memory to accelerate HGNNs, which naturally reduces data movement.

ChainPIM Versus REFLIP: We compare ChainPIM with the state-of-the-art ReRAM-based accelerator REFLIP specially designed for GCN. Although ChainPIM and REFLIP both employ ReRAM crossbar as the basic compute unit, ChainPIM outperforms REFLIP by 15.35× on average. The reasons are twofold. First, in addition to intervertex parallelism, ChainPIM further exploits unique interinstance and intermetapath parallelism of HGNNs. Second, ChainPIM additionally enhances vertex reuse by employing the locality-aware instance scheduling and more effectively improves hardware utilization through the adaptive crossbar allocation across different engines.

ChainPIM Versus MetaNMP: We also compare ChainPIM with the stae-of-the-art HGNNs accelerator MetaNMP. MetaNMP uses near-memory processing architecture and customized hardware for HGNNs. ChainPIM is 6.01× faster than MetaNMP on average. There are three main reasons for this. The first is that our crossbar-based processing performs the MVM operation, providing more computational parallelism than MetaNMP. Second, although near-memory processing and processing-in-memory both aim to address the irregular

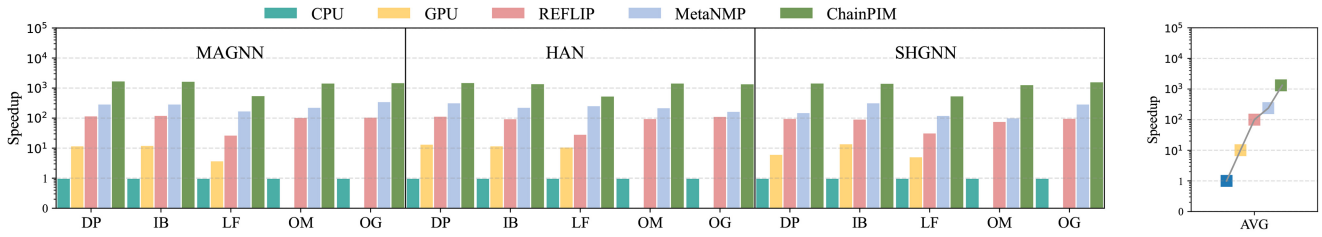

Fig. 11. Speedup of ChainPIM compared with CPU, GPU, REFLIP and MetaNMP. All results are normalized to CPU. OM and OG fail in running on GPU due to out of memory.

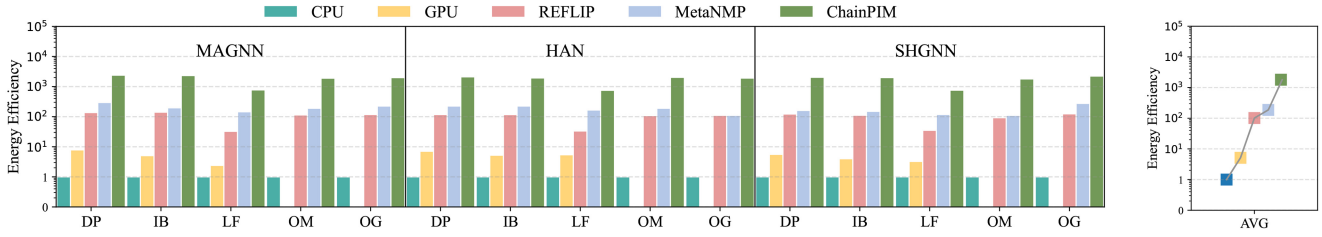

Fig. 12. Energy efficiency of ChainPIM, CPU, GPU, REFLIP and MetaNMP. All results are normalized to CPU.

and frequent memory accesses, processing-in-memory is better at reducing data movements, especially in HGNNs where high-dimensional features are frequently loaded and stored. Our R-chain based locality-aware scheduling extends this advantage. Third, ChainPIM considers instance-level, metapath-level, and vertex-level parallelism, thus fully exploiting the potential of parallelism in HGNNs and achieving a remarkable performance improvement.

C. Energy Efficiency

Fig. 12 shows the energy efficiency of ChainPIM compared with CPU, GPU, and MetaNMP [19]. It can be seen that ChainPIM consumes 1802× less energy than CPU and 331.75× less energy than GPU on average. This is due to processing-in-memory employed in ChainPIM, which significantly reduces data movement and performs low-energy MVM operations. Moreover, ChainPIM saves energy by an average of 18.49× compared to REFLIP. This is because ChainPIM reduces a large amount of repeated vertex loading through the locality-aware scheduling, thereby eliminating unnecessary ReRAM write overhead. Additionally, ChainPIM outperforms MetaNMP with 10.07× less energy consumption on average. The underlying reasons for this are twofold. First, computations in ChainPIM are performed where the data is stored, while MetaNMP is performed near the place where data is stored, indicating that ChainPIM has less data movement. Second, ChainPIM performs low-energy MVM operations efficiently, while MetaNMP performs computations element by element by integrated compute logic.

D. Memory Consumption Efficiency

Metapath instances result in many vertices being stored repeatedly, causing significant memory consumption. Especially for the large graphs where metapath instances increase exponentially, such as OGB-MAG [36], the memory consumption required in the model is much greater than the graph itself due to the hundreds of millions of metapath instances to store. Experimental results show that ChainPIM reduces memory consumption by 47.86% on average, as shown in Fig. 13. A reason behind this is that we design a new storage format for R-chain to avoid storing a large number of repeated vertices. In addition, with the pipelined parallelism between the R-chain processor and ReRAM, the R-chain's storage space will be released in time after it is aggregated, reducing the memory consumption.

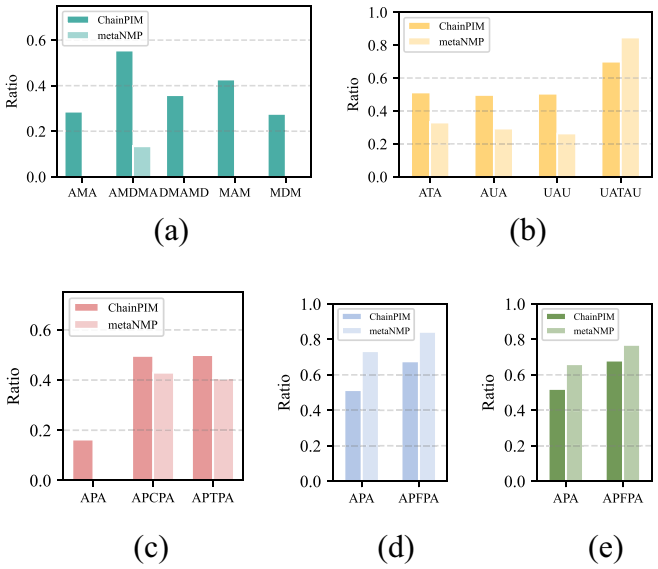

Fig. 13. Memory consumption reduction ratio between ChainPIM and MetaNMP. (a) IB. (b) LF. (c) DP. (d) OM. (e) OG.

Moreover, ChainPIM outperforms MetaNMP for most metapaths in IB, LF and DP. Although both systems generate instances dynamically, ChainPIM additionally reduces the repeated vertices. Notably, for metapaths in OM and OG, ChainPIM is mildly inferior to MetaNMP. This is because ChainPIM has more powerful parallelism and needs to store

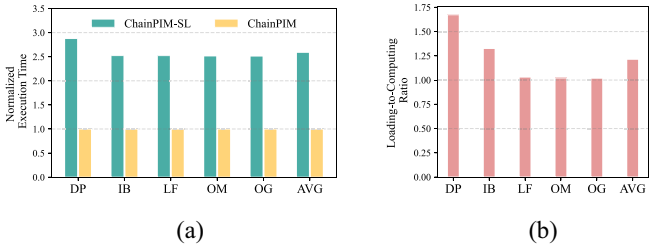

Fig. 14. (a) Performance comparison between ChainPIM and ChainPIM-SL and (b) loading-to-computing ratio.

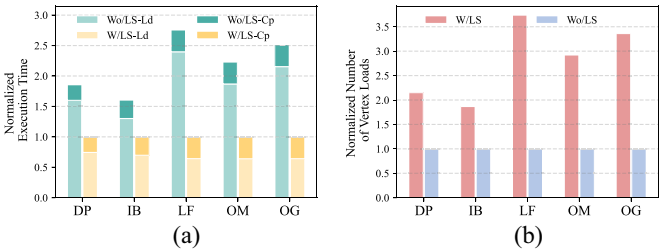

Fig. 15. (a) Execution times and (b) number of vertex data loads comparison between W/LS and Wo/LS. All results are normalized to W/LS.

more instances for parallel vertices. Sacrificing a small amount of memory to enhance performance proves to be cost-effective. Besides, heterogeneous graph in real world is typically dynamic, which changes slightly in a period of time. In this scenario, ChainPIM requires minor modifications to the R-chains each time, while MetaNMP has to regenerate the instances, indicating that ChainPIM offers better efficiency than MetaNMP when handling dynamic graph data.

E. Performance Breakdown of ChainPIM

1) Instances Storage Over R-Chain: In addition to reduce memory consumption, our instances storage method also provides the execution order of metapath instances in advance, allowing the crossbars to prefetch the vertex features. Fig. 14(a) demonstrates the performance of data prefetching. ChainPIM-SL is a version of ChainPIM that does not have the execution order of instances and loads vertex features naively to the crossbars. ChainPIM outperforms ChainPIM-SL by 2.59× on average and DP achieves the best-performance improvement of 2.87×. The reason behind this is that instance management stores the relationships between instances, allowing for efficient data prefetching. In addition, the performance achieved is highly correlated with the loading-to-computing ratio, as shown in Fig. 14(b). DBLP has the highest-loading-to-computing ratio and, therefore, delivers greater gains in data prefetching.

2) Instance Scheduling Over R-Chain: Fig. 15(a) presents the execution time of MAGNN inference for ChainPIM with (W/LS) and without (Wo/LS) our locality-aware scheduling. The normalized execution time is divided into computation time (W/LS-Cp and Wo/LS-Cp) and data load time (W/LS-Ld and Wo/LS-Ld). W/LS outperforms Wo/LS by 2.17× on average. This is because vertex features are fully reused in our locality-aware strategy, eliminating redundant vertex feature loading. As shown in Fig. 15(b), the performance

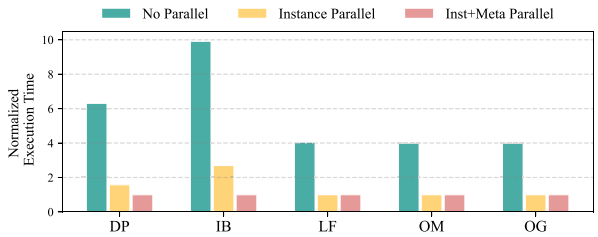

Fig. 16. Performance improvement in instance-level and metapath-level parallelism.

improvements achieved through the locality-aware scheduling order are closely related to the metapath correlation and the graph structure. LF has a relatively large number of metapath instances, and its metapaths have a relatively greater correlation. Therefore, the number of vertex data loads is reduced by 3.74×, obtaining a performance improvement of up to 2.76×.

3) Instance-Level and Metapath-Level Parallelism Exploiting: ChainPIM fully exploits instance-level and metapath-level parallelism. Fig. 16 shows the performance improvement from instance-level parallelism and metapath-level parallelism on five different datasets. We evaluate the normalized execution time in different configurations: 1) no instance-level and metapath-level parallelism (no parallel); 2) only employ instance-level parallelism (instance parallel); and 3) employ both instance-level and metapath-level parallelism (Inst+Meta parallel). Overall, instance-level and metapath-level parallelism boost the performance by 3.89× and 1.47× on average, respectively. The benefits obtained from these parallelism are closely related to the number of instances and metapaths. IB achieves the best-performance improvement because it has a maximum number of metapaths. Meanwhile, the number of instances corresponding to different metapaths is almost the same in IB, which allows the well-parallelized processing of metapaths to reduce execution time. However, LF, OM, and OG show slight improvement in metapath-level parallelism. The reason behind this is that the number of instances across different metapaths is extremely different, which leads to insignificant gains from metapath-level parallelism. When a metapath with a high number of instances is parallelized with a metapath with a low number of instances, the final time depends on the metapath with a high number of instances.

4) Sample-Based Crossbar Configuration: We investigate the performance and the crossbar utilization improvement from adaptively allocating the crossbar for different pipeline engines on five datasets. And we evaluate them in two configurations: 1) identical configuration that allocates the same number of crossbars for each engine (ChainPIM-Id) and 2) adaptive configuration using the sample-based approach proposed in Section IV-B (ChainPIM-Ad). As shown in Fig. 17(a), ChainPIM-Ad outperforms ChainPIM-Id by 2.10× on average. This is because the identical configuration of the three engines does not account for the imbalanced workload among them, leading to significant crossbar waste and degraded performance. In contrast, ChainPIM-Ad takes into

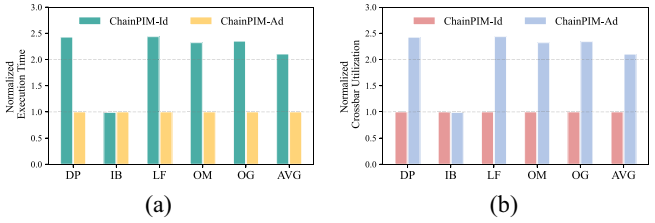

Fig. 17. (a) Normalized execution times and (b) crossbar utilization between ChainPIM-Ad and ChainPIM-Id. All results are normalized to ChainPIM-Id.

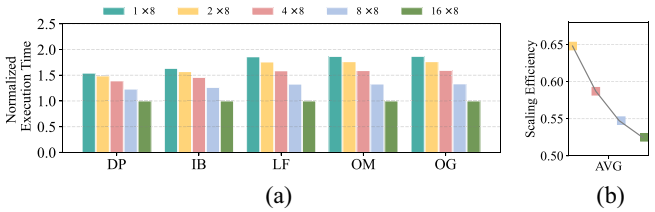

Fig. 18. (a) Normalized execution times and (b) scaling efficiency for various number of crossbars in CU.

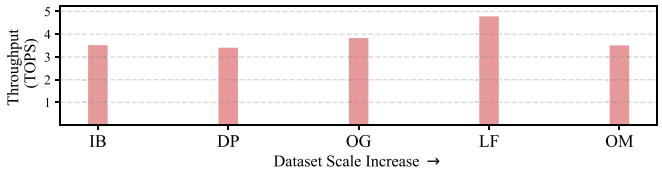

Fig. 19. Scalability for various scales of datasets.

account the number of metapaths and instances, crossbar size, etc., thereby enhancing hardware utilization through effective parallelism. As shown in Fig. 17(b), ChainPIM-Ad improves normalized crossbar utilization by 2.10× on average compared to ChainPIM-Id. All datasets show improved performance, except for IB, this dataset has a limited number of instances, rendering the workload too small to be effective for our ChainPIM architecture.

F. Scalability and Sensitivity Analysis

1) CU Size: We investigate the scalability for various numbers of crossbars in CU as shown in Fig. 18(a). The performance of ChainPIM exhibits good scalability as the number of crossbars increases. Since most of the feature dimensions are equal to or greater than 64, each CU is configured with 8 crossbars in a row, making the parallelism of crossbars fully exploited. However, the scaling efficiency decreases as the number of crossbars increases, as shown in Fig. 18(b). This is because as the number of crossbars further increases, the bottleneck of ChainPIM shifts from parallelism to communication. Therefore, we make a tradeoff between parallel efficiency and communication efficiency by setting the crossbars to 4×8.

2) Dataset Scales: We investigate the scalability of five datasets with various scales of metapath instances, namely, IB (0.009 GB), DB (1.83 GB), OG (83 GB), LF (111 GB), and OM (153 GB). Fig. 19 shows the throughput results of ChainPIM. Overall, the throughput of ChainPIM exhibits excellent scalability as the scale of datasets increases, achieving even higher throughput in larger datasets. Additionally, LF

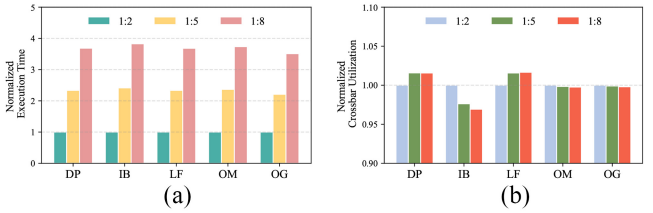

Fig. 20. (a) Normalized execution times and (b) crossbar utilization under different write-to-cost ratios, 1:2, 1:5, and 1:8. All results are normalized to 1:2.

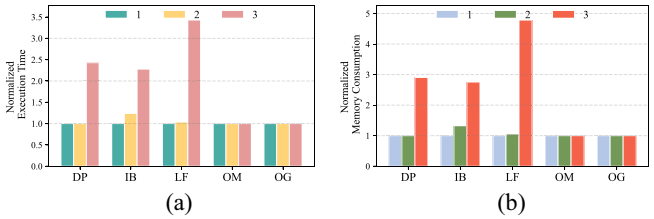

Fig. 21. (a) Normalized execution times and (b) memory consumption under different repetition factor $n = 1, 2, 3$. All results are normalized to $n = 1$.

achieves the best performance at 4.81 TOPS since it has a relatively larger potential for vertex reuse and a lower-loading-to-computing ratio, allowing more aggregation operations to execute in less time. However, OM does not achieve the higher performance expected because it has a relatively higher number of short instances. Therefore, it leaves more rows of the crossbar in idle, resulting in ordinary performance.

We investigate the effectiveness and scalability in adapting to various read-to-write cost ratios, namely, 1:2, 1:5, and 1:8. Fig. 20(a) shows normalized execution time under different read-to-write ratios. The degradation in system performance markedly less than the increase in write costs, indicating the effectiveness of our adaptive hardware allocation method in Section IV-B. Moreover, Fig. 20(b) illustrates the normalized crossbar utilization under different read-to-write ratios. The results indicate that ChainPIM maintains robust performance even as the read-to-write cost ratio increased to 1:8, further demonstrating the effectiveness and scalability of ChainPIM in adapting to various read-to-write cost ratios.

3) Repetition Factor n: Fig. 21 shows the normalized performance and memory consumption of ChainPIM under different repetition factor n. Repetition factor n represents a tradeoff between parallelism and vertex reuse. The number of instances for a vertex is usually much larger than the number of crossbars in PE, implying that parallelism of ReRAM is fully exploited while vertex reuse is not. Therefore, n primarily affects the reuse of vertices. As shown in Fig. 21(a), $n = 1$ is the best value for all the datasets. This is because small n can fully exploit vertex reuse, thereby reducing a large amount of vertex load. Large n results in low-vertex reuse and ReRAM cannot offer sufficient parallel elements for large n, leading to degraded performance. Besides, as shown in Fig. 21(b), memory consumption demonstrates similar pattern to performance. Setting $n = 1$ optimally reduces the repeated vertices, thereby achieving minimal memory consumption. Notably, OM and OG are not sensitive to n, this is because

they have only 2 metapaths and their instances within different metapaths have negligible shared vertices.

VI. Conclusion

In this article, we present ChainPIM, the first ReRAM-based processing-in-memory accelerator for HGNNs. ChainPIM is designed to significantly enhance the efficiency of HGNNs through three key innovations. First, we use R-chain to capture the parallelism and reuse relationship among metapath instances, enabling fast generation of efficient execution schemes. Second, an efficient storage method is designed for R-chain to reduce redundant vertices storage significantly. Third, we develop a specialized ReRAM-based architecture to pipeline the different types of aggregations in HGNNs, fully exploiting the multilevel parallelism. Experimental results demonstrate that ChainPIM outperforms CPU, GPU, and state-of-the-art accelerators for HGNNs in both performance and energy efficiency.

References

[1] Y. Wang et al., "GNNAdvisor: An adaptive and efficient runtime system for GNN acceleration on GPUs," in *Proc. USENIX Symp. Oper. Syst. Design Implement. (OSDI)*, 2021, pp. 515–531.

[2] M. Kwon, D. Gouk, S. Lee, and M. Jung, "Hardware/software co-programmable framework for computational SSDs to accelerate deep learning service on large-scale graphs," in *Proc. USENIX Conf. File Storage Technol. (FAST)*, 2022, pp. 147–164.

[3] Y. Huang et al., "Accelerating graph convolutional networks using crossbar-based processing-in-memory architectures," in *Proc. IEEE Int. Symp. High Perform. Comput. Architecture (HPCA)*, 2022, pp. 1029–1042.

[4] A. I. Arka, B. K. Joardar, J. R. Doppa, P. P. Pande, and K. Chakrabarty, "DARe: DropLayer-aware manycore ReRAM architecture for training graph neural networks," in *Proc. IEEE/ACM Int. Conf. Comput. Aided Design (ICCAD)*, 2021, pp. 1–9.

[5] H. Wang, Z. Cui, R. Liu, L. Fang, and Y. Sha, "A multi-type transferable method for missing link prediction in heterogeneous social networks," *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 11, pp. 10981–10991, Nov. 2023.

[6] C. Chen and Y.-Y. Liu, "A survey on hyperlink prediction," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 35, no. 11, pp. 15034–15050, Nov. 2024.

[7] A. C. M. Mancino, A. Ferrara, S. Bufi, D. Malitesta, T. Di Noia, and E. Di Sciascio, "KGTORe: Tailored recommendations through knowledge-aware GNN models," in *Proc. 17th ACM Conf. Recommender Syst.*, 2023, pp. 576–587. [Online]. Available: https://doi.org/10.1145/3604915.3608804

[8] J. Liu, Y. Wang, Z. Lin, M. Chen, Y. Hao, and L. Hu, "Natural language fine-tuning," 2024, *arXiv:2412.20382*.

[9] Y.-L. Zheng, W.-Y. Yang, Y.-S. Chen, and D.-H. Han, "An energy-efficient inference engine for a configurable ReRAM-based neural network accelerator," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 42, no. 3, pp. 740–753, Mar. 2023.

[10] H. Li, Z. Li, Z. Bai, and T. Mitra, "ASADI: Accelerating sparse attention using diagonal-based in-situ computing," in *Proc. IEEE Int. Symp. High-Perform. Comput. Archit. (HPCA)*, 2024, pp. 774–787.

[11] L. Zheng et al., "PhGraph: A high-performance ReRAM-based accelerator for hypergraph applications," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 43, no. 5, pp. 1318–1331, May 2024.

[12] H. Li et al., "ReSMA: Accelerating approximate string matching using ReRAM-based content addressable memory," in *Proc. 59th ACM/IEEE Design Autom. Conf.*, 2022, pp. 991–996.

[13] H. Li, H. Jin, L. Zheng, and X. Liao, "ReSQM: Accelerating database operations using ReRAM-based content addressable memory," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 11, pp. 4030–4041, Nov. 2020.

[14] D. Chen et al., "GraphFly: Efficient asynchronous streaming graphs processing via dependency-flow," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, 2022, pp. 1–14.

[15] Y. Huang et al., "Ready: A ReRAM-based processing-in-memory accelerator for dynamic graph convolutional networks," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 41, no. 11, pp. 3567–3578, Nov. 2022.

[16] 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.

[17] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 1263–1272.

[18] X. Fu, J. Zhang, Z. Meng, and I. King, "MAGNN: Metapath aggregated graph neural network for heterogeneous graph embedding," in *Proc. Web Conf.*, 2020, pp. 2331–2341.

[19] D. Chen et al., "MetaNMP: Leveraging cartesian-like product to accelerate HGNNs with near-memory processing," in *Proc. 50th Annu. Int. Symp. Comput. Archit.*, 2023, pp. 1–13.

[20] B. Hu, Z. Zhang, C. Shi, J. Zhou, X. Li, and Y. Qi, "Cash-out user detection based on attributed heterogeneous information network with a hierarchical attention mechanism," in *Proc. AAAI Conf. Artif. Intell. (AAAI)*, 2019, pp. 946–953.

[21] X. Geng, H. Zhang, J. Bian, and T. Chua, "Learning image and user features for recommendation in social networks," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, 2015, pp. 4274–4282.

[22] M. Yasunaga et al., "ScisummNet: A large annotated corpus and content-impact models for scientific paper Summarization with citation networks," in *Proc. AAAI Conf. Artif. Intell. (AAAI)*, 2019, pp. 7386–7393.

[23] D. Chen et al., "A general offloading approach for near-DRAM processing-in-memory architectures," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. (IPDPS)*, 2022, pp. 246–257.

[24] P. Chi et al., "PRIME: A novel processing-in-memory architecture for neural network computation in ReRAM-based main memory," *ACM SIGARCH Comput. Archit. News*, vol. 44, no. 3, pp. 27–39, 2016.

[25] A. Shafiee et al., "ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," *ACM SIGARCH Comput. Archit. News*, vol. 44, no. 3, pp. 14–26, 2016.

[26] L. Song, Y. Zhuo, X. Qian, H. Li, and Y. Chen, "GraphR: Accelerating graph processing using ReRAM," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, 2018, pp. 531–543.

[27] M. Hu et al., "Dot-product engine for neuromorphic computing: Programming 1T1M crossbar to accelerate matrix-vector multiplication," in *Proc. 53rd Annu. Design Autom. Conf.*, 2016, pp. 1–6.

[28] D. E. Knuth, *The Art of Computer Programming*, vol. 3. London, U.K.: Pearson Educ., 1997.

[29] D. Niu, C. Xu, N. Muralimanohar, N. P. Jouppi, and Y. Xie, "Design of cross-point metal-oxide ReRAM emphasizing reliability and cost," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, 2013, pp. 17–23.

[30] 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.

[31] M. O'Connor, "Highlights of the high-bandwidth memory (HBM) standard," in *Proc. Memory Forum Workshop*, vol. 3, 2014, pp. 1–25.

[32] R. Balasubramonian, A. B. Kahng, N. Muralimanohar, A. Shafiee, and V. Srinivas, "CACTI 7: New tools for interconnect exploration in innovative off-chip memories," *ACM Trans. Archit. Code Optim.*, vol. 14, no. 2, pp. 1–25, 2017.

[33] "Intel Xeon gold 5117 processor (19.25M cache, 2.00 GHz) specifications," Intel. 2024. [Online]. Available: https://www.intel.com/content/www/us/en/products/sku/122460/intel-xeon-gold-5117-processor-19-25m-cache-2-00-ghz/specifications.html

[34] "NVIDIA system management interface," Nvidia, 2012. [Online]. Available: https://developer.nvidia.com/nvidia-system-management-interface

[35] J. Gao, F. Liang, W. Fan, Y. Sun, and J. Han, "Graph-based consensus maximization among multiple supervised and unsupervised models," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 22, 2009, pp. 1–13.

[36] W. Hu et al., "Open graph benchmark: Datasets for machine learning on graphs," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp. 22118–22133.

[37] A. Sinha et al., "An overview of microsoft academic service (MAS) and applications," in *Proc. 24th Int. Conf. World Wide Web*, 2015, pp. 243–246.

[38] X. Wang et al., "Heterogeneous graph attention network," in *Proc. World Wide Web Conf.*, 2019, pp. 2022–2032.

[39] W. Xu, Y. Xia, W. Liu, J. Bian, J. Yin, and T.-Y. Liu, "SHGNN: Structure-aware heterogeneous graph neural network," 2021, *arXiv:2112.06244*.

Wenjing Xiao received the Ph.D. degree from the School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, China, in 2023.

She is a Research Assistant Professor with the School of Computer, Electronic and Information, Guangxi University, Nanning, China. Her Google Scholar Citations reached more than 450 with a H-index of 13. Her research interests include efficient artificial intelligence, edge intelligence, on device artificial intelligence, Internet of Things, and wireless network.

Jianyu Wang received the B.E. degree in mechanical engineering from the School of Mechanical Engineering, Jiangnan University, Wuxi, China, in 2022. He is currently pursuing the M.E. degree with the School of Computer, Electronics and Information, Guangxi University, Nanning, China.

His research interests include in-memory computing and graph neural networks.

Dan Chen received the Ph.D. degree from the Huazhong University of Science and Technology, Wuhan, China, in 2024.

He is currently the Research Fellow with the National University of Singapore, Singapore. His research interests focus on graph processing and processing-in-memory.

Chenglong Shi received the B.E. degree in computer science and technology from the School of Information Engineering, Xuchang University, Xuchang, China, in 2023. He is currently pursuing the M.E. degree with the School of Computer, Electronics and Information, Guangxi University, Nanning, China.

His research interests include edge computing, graph neural networks, and deep learning.

Xin Ling received the bachelor's degree in chemical engineering from the School of Chemistry and Chemical Engineering, Guangxi University, Nanning, China, in 2022, where he is currently pursuing the M.E. degree with the School of Computer, Electronics and Information.

His research interests include edge computing, Internet of Vehicles, and task offloading.

Min Chen (Fellow, IEEE) is a Full Professor with the School of Computer Science and Engineering, South China University of Technology, Guangzhou, China. He is also the Director of Embedded and Pervasive Computing (EPIC) Lab, Huazhong University of Science and Technology, Wuhan, China. His Google Scholar Citations reached more than 45 800 with a H-index of 98. His top paper was cited more than 4750 times.

Dr. Chen was selected as a Highly Cited Researcher from 2018 to 2023. He received the IEEE Communications Society Fred W. Ellersick Prize in 2017, the IEEE Jack Neubauer Memorial Award in 2019, and the IEEE ComSoc APB Oustanding Paper Award in 2022. He is the Founding Chair of IEEE Computer Society Special Technical Communities on Big Data. He is a Fellow of IET.

Thomas Wu received the Ph.D. degree in electrical engineering from the University of Pennsylvania, Philadelphia, PA, USA, in 1999.

In 1999, he was an Assistant Professor with the University of Central Florida, Orlando, FL, USA, where he was promoted to an Associate Professor in 2005 and a Professor in 2011. He also got his tenure in 2005. He was an ASEE Summer Faculty Fellow with Air Force Research Laboratory (AFRL) in Summer of 2009 and 2010. He was also appointed as the prestigious National Research Council Senior Research Associate with AFRL from 2010 to 2012. He is currently a Professor of Electrical Engineering with the School of Electrical Engineering, Guangxi University, Nanning, China.