MetaNMP: Leveraging Cartesian-Like Product to Accelerate HGNNs with Near-Memory Processing

Dan Chen*
cdhust@hust.edu.cn
Huazhong University of Science and Technology
Wuhan, China

Haiheng He*
hhh@hust.edu.cn
Huazhong University of Science and Technology
Wuhan, China

Hai Jin*
hjin@hust.edu.cn
Huazhong University of Science and Technology
Wuhan, China

Long Zheng*
longzh@hust.edu.cn
Huazhong University of Science and Technology
Wuhan, China

Yu Huang*
yuh@hust.edu.cn
Huazhong University of Science and Technology
Wuhan, China

Xinyang Shen*
xyshen@hust.edu.cn
Huazhong University of Science and Technology
Wuhan, China

Xiaofei Liao*
xfliao@hust.edu.cn
Huazhong University of Science and Technology
Wuhan, China

ABSTRACT

Heterogeneous graph neural networks (HGNNs) based on metapath exhibit powerful capturing of rich structural and semantic information in the heterogeneous graph. HGNNs are highly memory-bound and thus can be accelerated by near-memory processing. However, they also suffer from significant memory footprint (due to storing metapath instances as intermediate data) and severe redundant computation (when vertex features are aggregated among metapath instances). To address these issues, this paper proposes MetaNMP, the first DIMM-based near-memory processing HGNNs accelerator with reduced memory footprint and high performance. Specifically, we first propose a cartesian-like product paradigm to generate all metapath instances on the fly for heterogeneous graphs. In this way, metapath instances no longer need to be stored as intermediate data, avoiding significant memory consumption. We then design a data flow for aggregating vertex features on metapath instances, which aggregates vertex features along the direction of the metapath instances dispersed from the starting vertex to exploit shareable aggregation computations, eliminating most of the redundant computations. Finally, we integrate specialized hardware units in DIMM to accelerate HGNNs with near-memory processing, and introduce a broadcast mechanism for edge data and vertex features to mitigate the inter-DIMM communication. Our evaluation shows that MetaNMP achieves the memory space reduction of 51.9% on average and the performance improvement by 415.18× compared to NVIDIA Tesla V100 GPU.

CCS CONCEPTS

• **Hardware** → **Emerging technologies**; • **Computer systems organization** → **Architectures**.

KEYWORDS

Heterogeneous graph neural networks, cartesian product, near-memory processing

ACM Reference Format:
Dan Chen, Haiheng He, Hai Jin, Long Zheng, Yu Huang, Xinyang Shen, and Xiaofei Liao. 2023. MetaNMP: Leveraging Cartesian-Like Product to Accelerate HGNNs with Near-Memory Processing. In *Proceedings of the 50th Annual International Symposium on Computer Architecture (ISCA '23), June 17–21, 2023, Orlando, FL, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3579371.3589091

1 INTRODUCTION

Research on graph neural networks in computer architecture has mostly focused on homogeneous graph composed of only one type of vertices and edges [2, 4, 16, 25, 33, 34, 46, 49]. Yet, there are a large number of problems in the field of graph neural networks on heterogeneous graph, consisted of multi-types of vertices and edges, which have received little or no attention from computer architecture. Actually, heterogeneous graphs are more common in real life and have a richer ability to represent information than homogeneous graphs. For example, as shown in Figure 1, an academic graph consists of multiple types of vertices: author, paper, and conference. The edges between different types of vertices express

*All authors are with National Engineering Research Center for Big Data Technology and System, Services Computing Technology and System Lab, Cluster and Grid Computing Lab, School of Computer Science and Technology in Huazhong University of Science and Technology. Long Zheng is the corresponding author of this paper.

ISCA '23, June 17–21, 2023, Orlando, FL, USA.

ACM ISBN 979-8-4007-0095-8/23/06…$15.00
https://doi.org/10.1145/3579371.3589091

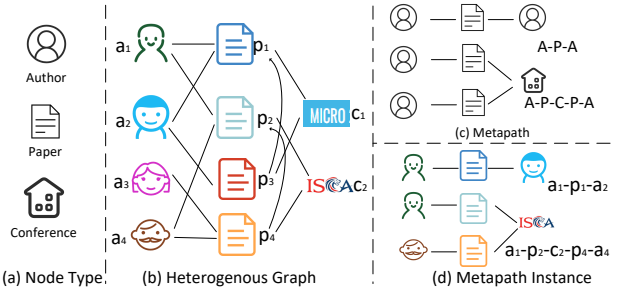

Figure 1: An illustrative example of a heterogeneous graph. (a) Three types of vertices (i.e., author, paper, conference). (b) An academic heterogeneous graph contains three types of vertices and three types of connections. (c) Two defined metapaths (i.e., Author-Paper-Author and Author-Paper-Conference-Paper-Author). (d) Two metapath instances

different information, e.g., '$a_1 - p_2$' indicates that the author a_1 wrote a paper p_2; '$p_2 - c_2$' indicates that the paper p_2 was published by conference c_2. Since HGNNs can capture powerful structural and semantic information in heterogeneous graphs, HGNNs appear to be particularly important and are core to a wide range of applications such as cybersecurity [20, 23], recommendation systems [17, 52], and social network analysis [51, 54].

To capture the rich structural and semantic information carried in heterogeneous graphs, metapath, an ordered sequence of vertex types and edge types describes the complex relationships between vertex types. For example, in Figure 1, *Author-Paper-Author* (APA) and *Author-Paper-Conference-Paper-Author* (APCPA) are metapaths expressing two different structural information. The APA metapath expresses two co-authors, and the APCPA metapath expresses two authors who published paper in the same conference. Most prevalent HGNNs capture the information based on metapath. They can be primarily divided into four stages: *Metapath Instance Matching* finds all instances that match the defined metapath in the heterogeneous graph; *Feature Projection* maps different dimensions of vertex and edge feature vectors to the same space; *Structural Aggregation* aggregates vertex features along metapath instance; and *Semantic Aggregation* aggregates the semantic information involved in all metapath instances. This is very different from graph neural networks in homogeneous graphs that aggregate information directly based on neighbor vertices. HGNNs have more complex aggregation along metapath. Existing software systems for accelerating HGNNs suffer from high memory footprint and severe redundant computations due to the following reasons.

First, multiple metapath instances can be generated starting from a vertex, and the number of instances is explosive compared to the number of vertices in the graph. Metapath instance matching is very time-consuming. It is usually treated as a pre-processing phase, where all metapath instances are stored to be used in the structural aggregation and semantic aggregation. This results in significant memory consumption. Our statistical study shows that the memory consumption for storing metapath instances is on average 239.84× larger than the graph data itself. Such high memory consumption limits the applicability of HGNNs on large graphs. As the growth rates of graph dataset sizes exceed the memory capacity, it becomes critical to develop metapath instance matching that does not require large memory consumption.

Second, vertex features are aggregated along the metapath instance, leading to severe redundant computations. Specifically, in the structural aggregation, features of all vertices within each metapath instance are aggregated. Since there are multiple same vertices among metapath instances, this results in repeated aggregation for these metapath instances.

Moreover, HGNNs are highly suffering from memory bottlenecks. Excluding the pre-processing metapath instance matching, structural aggregation takes up to 83.56% of the inference time, and it involves a large number of irregular memory accesses in aggregating vertex features along the metapath instance. This phase has low arithmetic intensity and therefore its bottleneck is limited by memory. Also, pre-processing metapath instance matching travels irregularly through the graph to access the edge data. It is similarly memory-bound. This motivates us to overcome the memory bottleneck of HGNNs by using *near-memory processing* (NMP), which offers high memory bandwidth, low data access latency, and low energy consumption [6, 10, 26, 27, 31, 53].

To tackle these challenges, in this paper, we propose MetaNMP, a new design to overcome the large memory consumption caused by storing metapath instances, eliminate most of the redundant computations, and use NMP to accelerate HGNNs. First, matching all metapath instances can be quickly performed by a cartesian-like product of vertices on their sets of different types of neighbors. This drives our cartesian-like product paradigm to efficiently generate metapath instances on the fly. Therefore, the pre-processing metapath instance matching is no longer needed, and the significant memory consumption caused by storing metapath instances is also avoided. Second, most of redundant computations across metapath instances arise from the fact that these instances are derived from the same vertex. Instead of aggregating each metapath instance independently, we carefully consider the data flow of aggregation along the direction of the metapath instances they derive from. This effectively removes these redundant computations.

For the near-memory acceleration of MetaNMP, we exploit DIMM- and rank-level parallelism in DRAM memory systems. We design specialized hardware units in DIMM-level to implement cartesian-like products for generating metapath instances and exploit shareable aggregation computations among metapath instances. We also integrate processing units in rank-level to aggregate vertices features. Additionally, both metapath instances generation and structural aggregation involve a large number of irregular memory accesses. The former involves irregular edge data accesses, and the latter involves irregular vertex feature accesses. This causes frequent communication among DIMMs. We further introduce broadcast mechanism between DIMMs in a regular manner to alleviate the frequent communication.

Our main contributions are summarized as follows:

- We investigate the problems of high memory footprint and redundant computation in HGNNs.
- We propose an efficient cartesian-like product paradigm for generating metapath instances on the fly and reducing redundant computations in a careful data flow aggregation manner.

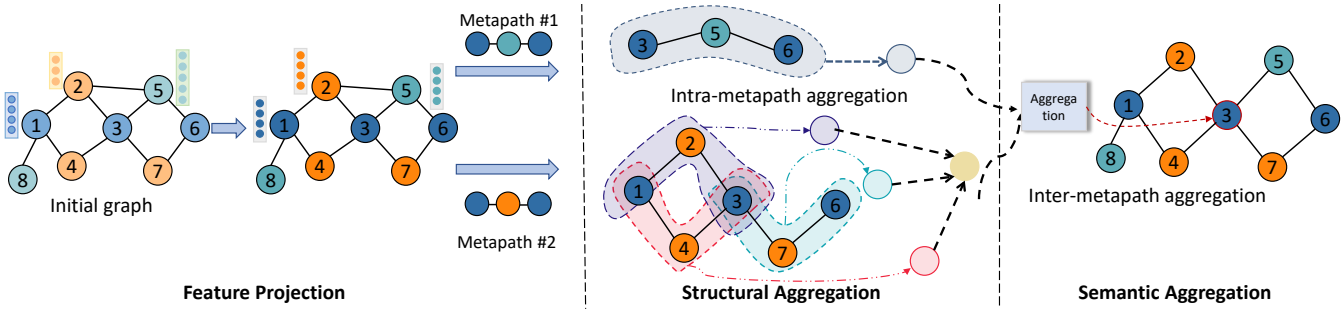

Figure 2: Illustration of the HGNNs model. As a pre-processing phase, metapath instance matching is excluded.

- We design a DIMM-based accelerator for near-memory processing of HGNNs and use a broadcast mechanism to alleviate inter-DIMM communication.
- Experimental results show that MetaNMP achieves the memory space reduction of 51.9% on average and the performance improvement by 415.18× compared to the NVIDIA Tesla V100 GPU.

2 BACKGROUND AND MOTIVATION

2.1 Heterogeneous Graph Neural Networks

Unlike traditional homogeneous graphs, where there is only one type of vertices and one type of edges, the heterogeneous graph usually contains multiple types of vertices and edges, denoted G = (V, E, V_t, E_t). V is the set of vertices. E is the set of edges. V_t is the set of vertex types, and E_t is the set of edge types. An important concept in heterogeneous graphs is the metapath, which is defined as a fundamental path in the form of $P = V_1 \xrightarrow{R_1} V_2 \xrightarrow{R_2} \dots \xrightarrow{R_L} V_{L+1}$, with $V_1V_2 \dots V_{L+1}$ representing a vertex-type sequence, $R_1R_2 \dots R_L$ representing an edge-type sequence, and L representing the length of P, expressing a composite of multiple relationships.

As shown in Figure 2, HGNNs usually consist of several phases. First, since the features of different types of vertices and edges are in different dimensional spaces, *Feature Projection* transforms features into the same space. Then, to capture structural information, *Structural Aggregation* aggregates vertex features within metapath intances. There are two types of aggregation in this phase. One is intra-instance aggregation, and the other is inter-instance with the same metapath for each vertex. Afterwards, *Semantic Aggregation* aggregates the semantic information revealed in all metapaths, i.e., aggregating the results of different metapaths for each vertex.

We summarize the characteristics of HGNNs as follows:

- *Distinct Feature Dimensions.* Different types of vertices and edges in a heterogeneous graph have different dimensions on their features.
- *Metapath-based Aggregation.* Traditional GNNs directly aggregate information based on the vertex's neighbors, while HGNNs leverage metapath instances to aggregate richer structural and semantic information.
- *Multi-level Aggregation.* Traditional GNNs contain only one aggregation and one combination in each layer. HGNNs include structural (intra-metapath) and semantic aggregation (inter-metapath).

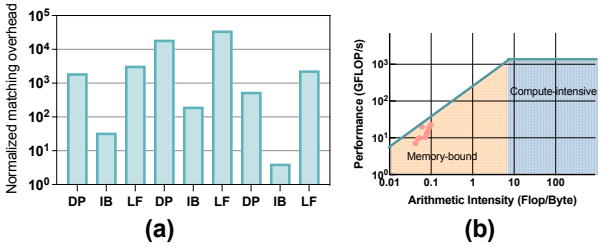

Figure 3: Performance characteristics on pre-processing metapath instance matching. (a) Execution time of metapath instance matching compared to the entire HGNNs inference time on NVIDIA Tesla V100. (b) Roofline of metapath instance matching on Intel Xeon Gold 5117 CPU

Time-consuming Metapath Instance Matching. Previous researches treated metapath instance matching as a pre-processing phase, performed on the CPU side, without considering its high time overhead. As shown in Figure 3(a), this phase is 8129.12× longer than the total inference time (the sum of three pahses as shown in Figure 2). They reuse metapath instances for each epoch in the training, thus this high cost can be amortized. However, it is intolerable in real-time inference. Particularly in real scenarios, graph data is continuously changed, which requires re-running the metapath instances matching when the graph changes. In this work, although we target at the large memory consumption of metapath instances, we also incidentally address the high overhead of metapath instance matching to make HGNNs more practical in real-world scenarios.

Memory Bottlenecks in HGNNs. We investigate the performance characteristics of HGNNs based on the roofline model [47], which is often utilized to deliver performance estimates for applications on architectures. Since GPU is usually used to accelerate HGNNs, we conduct the performance characteristics using the state-of-the-art HGNNs framework PyG [11] on NVIDIA Tesla V100. Figure 4(b) shows the roofline model for different phases of HGNNs inference. Because the large heterogeneous graphs are out of memory in the GPU, we only use small heterogeneous graphs. We can see that both structural aggregation and semantic aggregation are memory bottlenecks, except for the feature projection being compute-intensive. Figure 4(a) further decomposes the inference time of HGNNs. Structural aggregation occupies 83.56% of the overall inference time. Structural aggregation and semantic aggregation together occupy 91.85%. In addition, Figure 3(b) shows

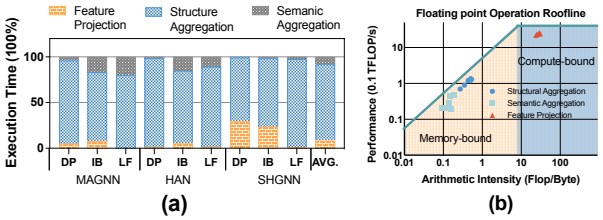

Figure 4: Performance characteristics on inference of HGNNs. (a) Execution time breakdown. (b) Roofline of reference on NVIDIA Tesla V100

Table 1: Comparison of memory consumption required for graph data and metapath instances

	DP	IB	LF	OM	OG
Graph Data	0.20GB	0.81GB	0.11GB	5.6GB	3.2GB
Instances	1.88GB	0.008GB	126GB	103GB	83GB

the roofline model of pre-processing metapath instances matching in the CPU. We can see that this phase is also limited by memory bottlenecks. In summary, the main time consumption of HGNNs inference is limited by memory, and near-memory processing to solve memory bottlenecks is urgently needed.

2.2 Problem of Existing Solutions

In HGNNs inference, the key is to aggregate the structural and semantic information involved in the metapath instances. However, the number of metapath instances is explosive compared to the number of vertices, where a small fraction of vertices may constitute a surprising number of instances. Besides, metapath instances contain multiple same vertices among them, which causes redundant aggregation. To demonstrate these problems, we conducted experiments on three HGNNs models (HAN, MAGNN, and SHGNN), based on the state-of-the-art framework PyG [11], using various types of datasets. Table 1 shows the memoy comsuption of metapath instances compared with graph data itself. Figure 5 shows the ratio of the redundant computations for MAGNN when vertex features within the metapath instance are aggregated.

Large Memory Consumption. Storing metapath instances is used in structural aggregation and semantic aggregation. Unfortunately, the number of metapath instances is explosive, as we describe this problem using the example in Figure 6(a). In this example, there are three types of vertices, A, B, and C, and we define the metapath as A-B-A. There are three vertices of the type A, i.e., ②, ④, ⑦, three vertices of the type B, i.e., ①, ③, ⑥. The number of metapath instances starting with the vertex ② is 5, i.e., ②-①-②, ②-①-④, ②-③-②, ②-③-④, and ②-③-⑦, and the total number of instances in the example is 14 for A-B-A. When the graph contains more types of vertices and longer metapaths are defined, the number of metapath instances grows exponentially. As shown in Table 1, the memory consumption for storing metapath instances is on average 239.84× larger than the graph data itself. Such large memory consumption limits the applicability of HGNNs on large graphs.

Redundant Computation. When vertex features within the metapath instances are aggregated, there are a large number of redundant computations among metapath instances. We illustrate

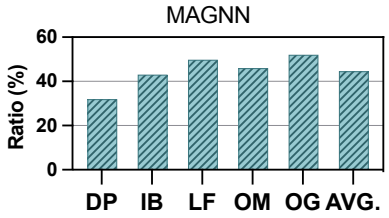

Figure 5: Ratio of redundant computations in MAGNN

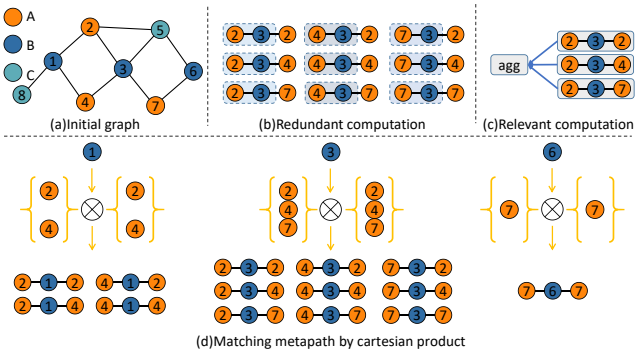

Figure 6: An example to explain metapath instances. (a) A heterogeneous graph. (b) Redundant computation among metapath instances. (c) Relationship among metapath instances. (d) Generating metapath instances using cartesian-like product

this using the example of Figure 6(b). There are 5 metapath instances starting at vertex ② for metapath A-B-A, three of which are ②-③-②, ②-③-④, and ②-③-⑦. These three instances contain two same vertices (②-③). In other words, all these three metapath instances need to aggregate the feature vectors of vertices ② and ③. As shown in Figure 5, the redundant computations are up to 44.56% in MAGNN. These redundant computations greatly limit the performance of HGNNs. Even worse, these vertex features are accessed irregularly when aggregated along metapath instance. Thus, it is difficult to hit in cache for the feature vectors, which results in expensive memory accesses.

3 OVERVIEW OF OUR SOLUTION

In this section, we present our intuition and main ideas of addressing memory consumption of metapath instances storage, redundant computations, and memory bottlenecks of HGNNs.

3.1 Metapath Instance Generation via Cartesian Product

The process of metapath instances matching can be transformed into cartesian products[1] between sets of different neighbor types of vertices. We illustrate this with an example that searching for all metapath A-B-A instances converts to cartesian-like products, as shown in Figure 6(d). We take the center vertex of type-B as the core. The type-B vertices contain ①, ③, ⑥. For the vertex ③, we take the

[1]Given any two sets X and Y, form an ordered pair using the elements in X as the first elements and the elements in Y as the second elements, the set consisting of all such ordered pairs is the cartesian product of sets X and Y.

vertex ③ as a fixed vertex, and then use the cartesian-like product of the type-A neighbors (left side of the type-B) with the type-A neighbors (right side of the type-B) to generate all A-B-A instances centered at vertex ③. For the other type-B vertices ① and ⑥, they have a similar process, as shown in Figure 6(d). The cartesian-like product approach brings a simple yet effective way for metapath instance generation. It can generate metapath instances directly at runtime based on the set of corresponding neighbor types of vertices to guide structural and semantic aggregation. This gives us an opportunity to avoid significant memory comsumption for storing metapath instances.

Above description only gives an illustration of generating metapath instances that contains three vertex types using cartesian-like product. There exist longer defined metapaths that contain more vertex types. Hence, we decompose the metapath and then transform it into multiple cartesian-like products. Specifically, for a given metapath $V_1V_2V_3...V_{L+1}$, we first perform a cartesian-like product on $V_1V_2V_3$, and the result, referred to as O, represents instances of sub-metapaths. To generate complete metapath instances, we treat the result O as a new type of vertex, and the metapath instance matching of $V_1V_2V_3...V_{L+1}$ is equivalent to $OV_3V_4...V_{L+1}$. This process can be repeated until the entire metapath matching is completed.

3.2 Relevant Metapath Instance Awareness

In the structural aggregation phase, there are two types of aggregation: intra-instance and inter-instance aggregation with the same metapath. In this subsection, we describe how to eliminate redundant computations for intra-instance aggregation and achieve efficient aggregation of inter-instance.

Redundant Computation Elimination. When a vertex in the defined metapath has multiple type-matched neighbors, it scatters out multiple metapath instances that exist redundant computation among them. For example, ②-③ is sub-metapath instance for metapath A-B-A in Figure 6. The vertex ③ has three type-A neighbors, so it can generate three instances, all of which contain redundant computations for aggregating the vertices ② and ③. If we aggregate each instance independently, it is difficult to capture these redundant computations, requiring significant detection overhead.

To efficiently eliminate redundant aggregation of these relevant instances, we use a simple yet effective relevant instance-aware approach where each instance can be naturally exploited for shareable computation during metapath instance generation. We inspect all dependencies (a dependency contains at least two vertices) starting from vertices of the first type in the metapath, and vertices on a dependency are aggregated directly. Once multiple sub-dependencies are scattered from a dependency, that is, multiple instances are generated. The previously aggregated result can be shared by these metapath instances. We further illustrate this with the example in Figure 6. At the beginning, we inspect all dependencies along the vertex ②. The vertex ③ is a path along the vertex ② out of the match type (③ and ② form a dependency), and we complete the vertex aggregation on them. When we then match vertex ③'s type-matched neighbor vertices along the metapath, we find that it has multiple type-matched neighbors, which results in multiple metapath instances and brings shareable aggregation computation

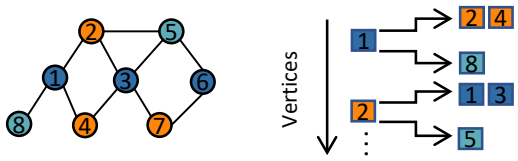

Figure 7: Optimized graph layout. Different neighbors are stored separately, such as vertex 1, which has two types of neighbors, vertex 2 and vertex 4 are stored together, and vertex 8 exists separately.

opportunity. The result of previous aggregation is then shared to these metapath instances.

Instance-driven Aggregation. Existing methods construct subgraphs to represent relevant metapath instances for each vertex by pre-processing. They ignore the information carried by the instance itself. The first vertex of each meta-path instance implicitly indicates which vertex they belong to. Therefore, instead of aggregating these related metapath instances using the gather-centered approach based on subgraphs, it is more efficient to drive the aggregation by metapath instance itself.

3.3 Near-Memory Processing

Software-only implementations of the above methods, despite their significant performance gains over existing solutions, still suffer from a high runtime overhead (see Section 5.4). First, it needs to traverse the graph with irregular access of edge data on runtime. Second, it needs to detect shareable aggregation computations among metapath instances. Besides, data-dependency limits instruction-level parallelism because these instructions depend on the result of the previous instruction. The high runtime overhead of a software-only implementation weakens the performance improvement of our approach, which motivates our NMP-based hardware design.

NMP is not the only potential solution for HGNNs. We select it because NMP is the most promising technique to overcome the memory bottleneck beyond other solutions. NMP can provide higher memory bandwidth, lower access latency, and lower energy consumption. Meanwhile, both metapath-based aggregation and metapath instance matching are memory-bound and occupy the major computation time of HGNNs, so NMP is naturally a suitable hardware solution for HGNNs. In our NMP-based designs, structural aggregation, semantic aggregation, and metapath instance matching are offloaded to the NMP to be accelerated by customized hardware, while feature projection is performed by the CPU.

4 METANMP ARCHITECTURE

4.1 Optimizing Graph Layout

Since generating metapath instances on the fly performs cartesian-like product operation with sets of different types of neighbors, the graph layouts can be improved to avoid expensive checking on vertex's neighbor type. Specifically, different types of vertex neighbors can be separated in the graph placement. By this way, vertex's different types of neighbors can be accessed directly on the graph layouts for cartesian-like product operation without the need to check all the vertex's neighbor type one by one. Such a graph representation is shown in Figure 7, where each vertex stores different

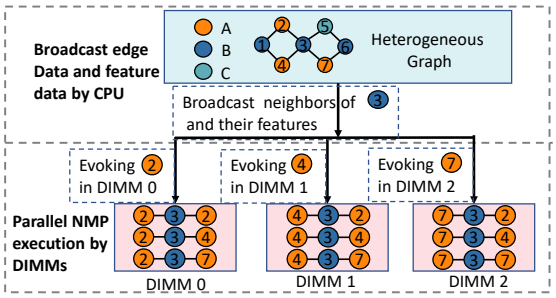

Figure 8: Broadcast mechanism of metaNMP on DIMMs

types of neighbors separately. This graph representation requires only minor modifications based on existing graph representations such as CSR.

4.2 Broadcast Mechanism

Although DIMM-based NMP is promising to overcome memory bottleneck of HGNNs, irregular edge data traversal and vertex feature aggregation along metapath instance yield inevitable communication among DIMMs. This would become the bottleneck of the DIMM-based NMP system to accelerate HGNNs. Inspired by the inter-DIMM broadcast approach [41], we propose a broadcast-based method, which broadcasts edge data and vertex features regularly to induce metapath instace generation and features aggregation along instance. As illustrated in Figure 8, we implement instance generation of metapath A-B-A and feature aggregation by broadcasting method with the graph stored in multiple DIMMs. The CPU first traverses each vertex in type-B as follows. Consider the vertex ③ as an example. It reads all the type-A neighbors of the vertex ③, and evokes the DIMMs where these type-A neighbor vertices are located to be responsible for generating metapath instances starting with them as well as completing feature aggregation. The CPU then broadcasts the type-A neighbor data of the vertex ③ to the DIMM. Note that the CPU will also broadcast these vertex features at the same time when it broadcasts these edge data. With this broadcast approach, data is shared regularly by DIMMs, and the communication bottleneck among DIMMs can be mitigated.

The broadcast phenomenon exists physically in current supported point-to-point communication [37, 41]. When the CPU transfers data through the bus, it physically changes the voltage level of the entire bus. In other words, it must charge the terminal capacitance of every DIMM on the bus. Once the charging process is completed, each DIMM sees the transferred data at its local terminal. Therefore, achieving broadcast among DIMMs only requires utilizing an existing physical mechanism, which primarily requires several modifications: 1) Broadcast command is added to the existing instruction format; 2) The DIMM buffer chip increases the functions of converting broadcast commands to regular DDR commands; 3) Modification on memory controller's Finite State Machines is added to consider the timing constraints of broadcast commands.

Note that not all data is broadcasted. For example, in Figure 8, when the metapath is C-A-B and there is only one type-C vertex ⑤ for type-A vertex ②, this means that only one DIMM needs the data. Therefore, in this case, normal communication can be employed. Furthermore, in multi-channel systems, the CPU broadcasts data one by one for each channel. Therefore, we do not need to broadcast data to other channels when the data is only required by a single channel. In our experiments, we only use the broadcast mechanism when there are at least two DIMMs within the channel that require the same data. Otherwise, normal communication is utilized.

4.3 Hardware Architecture

The overall architecture of MetaNMP is shown in Figure 9(a). We additionally include two modules on DIMMs to accelerate HGNNs. The hardware modifications of the added modules are limited to the buffer chip on each DIMM without any modifications to the commodity DRAM device. The DIMM-MetaNMP module at DIMM-level is responsible for generating metapath instances using cartesian-like product and exploits shareable aggregation among metapath instances. The rank-AU module in each rank performs aggregation, which exploits rank-level bankwidth.

DRAM Chips (Figure 9(b)). At the DRAM chip level, we reserve a part for storing the aggregation results for each metapath instance, which does not involve modification of the internal circuit design for the DRAM device. Because the metapath instance is generated gradually in a cartesian-like product manner, this aggregation results region enables DIMM-MetaNMP module to directly allocate the storage of aggregation results for each metapath instance in DRAM without the help of the host. In our experiments, we observe that 128MB per DIMM for aggregation results is sufficient, which takes up only 1.56% of the memory space.

Cartesian-like Product Unit (CarPU). As shown in Figure 9(d), CarPU contains two buffer queues (a type-1 queue and a type-3 queue) and a specific type-2 vertex register, used to perform a cartesian-like product for generating metapath instances. Under the control of the ctrl logic, one instance can be generated per cycle. The CarPU can be configured to perform the standard cartesian product by disabling the specific type-2 vertex register through an AND gate. Additionally, when the neighbors of one type exceed the capacity of the type-1/type-3 buffer queue, we can solve it by decomposing the cartesian-like product into multiple completions.

Reusable Computation Exploitation Unit (RCEU). RCEU is responsible for detecting reusable computations when CarPU generates metapath instances. Except for the first vertex in the type-3 buffer queue, other vertices in the queue have reusable computations with this first vertex. Therefore, as shown in Figure 9(e), the RECU takes the vertex's sequential number in the type-3 queue as input and shifts it to the right by one bit with a shifter to determine if a reusable computation exists, i.e., if the result is not 0, reusable computation exists. The RCEU can be disabled by setting the mode register.

DIMM-MetaNMP (Figure 9(c)). When the NMP-inst decoder receives an instruction from the host-side memory controller, the rank-level instruction is dispatched to the corresponding rank. DIMM-MetaNMP decodes other instructions and executes them to generate metapath instances and exploit reusable computations. The edge data stored in the edge buffer is sent to the CarPU to generate metapath instances, and the RCEU monitors the instance generation process for detecting redundant computations. Once an instance is generated, the controller stores it in the metapath

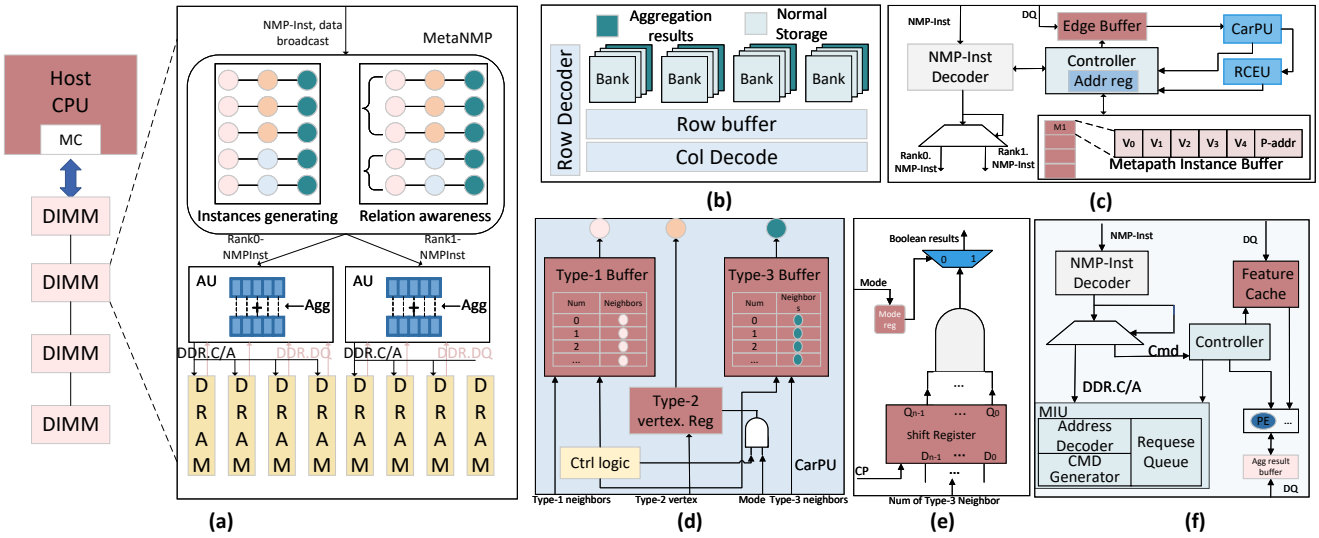

Figure 9: (a) Architecture of MetaNMP; (b) DRAM chip; (c) DIMM-MetaNMP is hardware integrated at the DIMM level for generating and sensing metapath instances; (d) Cartesian-like product unit (CarPU); (e) Reusable computation exploitation unit (RCEU); (f) Aggregation unit (AU)

instance buffer, where each item stores up to five vertices, due to the fact that metapaths are typically under 5 in length. For the longer metapath, two items can be used to store an instance. Each item also contains a physical address to enable fast access for aggregation results. This physical address is allocated by the controller based on an address register. Meanwhile, the controller generates aggregation instructions and copy instructions for reusable computation results to rank level.

Aggregation Unit (AU). One AU per rank is configured, as shown in Figure 9(f), to exploit the rank-level bandwidth for feature aggregation. When the decoder receives a NMP-inst, standard DDR C/A commands will be sent to the *memory interface unit* (MIU), while feature aggregation commands and copy commands for reusable computation results will be sent to the controller inside the AU. For feature aggregation commands, the controller requests the data of aggregation results by sending its physical address (carried on NMP-inst) to memory interface unit. The requested aggregation results are stored in buffer. Then the controller aggregates the vertex features stored in feature cache with aggregation results through *processing elements* (PEs), which consists of multipliers and adders. While for copy commands, the controller only needs to read the aggregation result of the reused instance and then write it to the aggregation result of another instance.

4.4 Programming Interface and Workflow

MetaNMP supports the virtual memory system. The host reads the edge and feature data of the vertices, and logical-to-physical address translation is performed at the host side. The metapath instance buffer stores the physical address of aggregation result for each instance, thereby enabling direct access without requiring translation. For memory layout, MetaNMP reserves a specific part of memory space for storing aggregation results, while allowing flexible placement of other data. The virtual memory system

	1 bit	32 bits	46 bits
Memory Mode	Mode(0)	Addr	Other cmd

	1 bit	4 bits	32 bits	4 bits	32 bits	6 bits
NMP Mode	Mode(1)	Op	Addr/Data	Mask	Addr/Data	To be mined
ConfigSize	1	0000	\	\	Feature_length	\
Evoke	1	0001	Vertex Num	\	Feature_addr	\
Broadcast	1	0010	\	Mask	Broadcast_addr	\
Broadcast_core	1	0011	Vertex Num	Mask	Broadcast_addr	\
Aggregate	1	0100	Vertex Num	\	Agg_addr	\
Inter_instance_agg	1	0101	Vertex Num	\	Output_addr	\
Copy	1	0110	Agg_addr	\	Dst_addr	\
ConfigWeight	1	0111	\	\	Weight	\
Inter_path_agg	1	1000	V_{path1}_addr	\	V_{path2}_addr	\

Figure 10: MetaNMP instruction format

only ensures that both features of a vertex and its final output are allocated completely within the same rank for efficient aggregation.

MetaMMP adopts heterogeneous computational programming models to enable NMP. The NMP instruction is explicitly indicated in the code to facilitate offloading it to NMP side. Figure 10 shows the instruction format of MetaNMP. Using the example of metapath A-B-C to complete instance generation and feature aggregation as shown in Figure 11, we walk through the workflow of MetaNMP. Initially, memory is allocated to the heterogeneous graph and the output result of each vertex. Since all vertices have the same feature dimension after feature projection, we directly configure the feature size to the controller in rank-AU module. For each type-2 vertex, the host first reads its type-1 neighbors and subsequently sends *evoke* instructions, which carries the vertex number of the neighbor and start physical address of the feature, to the DIMM where the neighbor is stored. The DIMM-MetaNMP module feeds the vertex number to the CarPU's type-1 queue, and the rank-AU module reads the features based on the start address to the feature cache. The host then broadcasts the type-2 vertex number and feature

```
// Initialization
Type-1, Type-2, Type-3 = A, B, C     //MetaPath: A-B-C
G, output=initia(HeterGraph)
//  NMP Kernels
CartesianProduct( G, Type-1, Type-2, Type-3, output){
    ConfigSize(G.vertexFeaturelength)
    Foreach V_Type-2 in G.getVertex(Type-2) do{
        Type-1_vertices=[]
        Foreach V_Type-1 in V_Type-2.neighbors(Type-1) do
            Evoke(V_Type-1)
            Type-1_vertices.add(V_Type-1)
        Broadcast_core(V_Type-2)
        Broadcast (V_Type-2.nerghbors(Type-3))
        Foreach V_Type-1 in Type-1_vertices do
            Inter_instance_agg(V_Type-1, &output[V_Type-1])
 }}
```

Figure 11: MetaNMP cartesian-like product example code

data to the evoked DIMMs using *broadcast_core* instruction. CarPU configures its vertex number to the specific register. Subsequently, the host broadcasts the type-3 neighbors and their feature data to these DIMMs using *broadcast* instruction. The type-3 neighbors are configured to the CarPU's type-3 queue, and the feature data are stored in the Rank-AU's feature cache. Then DIMM-MetaNMP starts generating metapath instances.

Regarding feature aggregation, the DIMM-MetaNMP controller directly generates the *aggregate* instruction, which carries the aggregated vertex number and the physical address of the aggregation result. For reusable computation, DIMM-MetaNMP sends a *copy* instruction containing the reusable aggregation result address and the destination address. Once the NMP side has completed the instance generation and feature aggregation within the instance, the host side sends an *inter_instance_agg* request to ask the NMP side to aggregate all the instance results of the given vertex to the output results. Subsequently, the controller in DIMM-MetaNMP generates *aggregate* instructions based on the relevant instances in the metapath instance buffer and sends them to the rank-AU to complete the aggregation process. When multiple metapaths exist, different metapath results for each vertex may need to be multiplied by the corresponding weight. MetaNMP configures rank-AU with the *configweight* instruction. Then the results of different metapaths are aggregated by *inter-path-agg* instruction.

We discuss exception handling in two cases. Firstly, when the program is suspended or preempted by other tasks, the NMP side can continue with previous offloaded task without affecting normal memory access. This is because aggregation results are stored in a reserved area. The program can resume execution after recovery. Secondly, if program execution encounters errors, correctness is guaranteed by recomputing vertices that were being computed when the exception occurred.

4.5 Generality for Other Application Domains

MetaNMP targets complex computations in HGNNs. Many applications in various domains are simple variants of HGNNs and can be accelerated by MetaNMP, including traditional GNNs, graph analysis, and essential sparse matrix multiplication for machine learning [19]. MetaNMP adopts a cartesian-like product computing paradigm, and these simple variant applications can be transformed into such paradigm to benefit from MetaNMP. For example, the aggregation of traditional GNNs can be considered as the cartesian product of the vertex and its neighbors. These computations are usually data-dependent, and thus limited by memory access. MetaNMP can help overcome memory bottlenecks, and our broadcast mechanism can also alleviate their inter-DIMM communication.

Table 2: Simulation parameters and configuration

Host Processor	16 out-of-order cores, 2.5GHz, 32KB L1 cache, 256KB L2 cache, 16MB LLC cache
DRAM Module	DDR4-2400MHz 8GB, 64GB total size, 4 Channels × 2 DIMMs × 2 Ranks, 4KB row buffer size, FR-FCFS
DRAM Timing Parameters	tRCD=16, tCL=16, tRP=16, tRC=55, tRRD_S=4, tRRD_L=6, tFAW=26, tCCD_S=4, tCCD_L=6, tBL=4
NMP Configuration	32KB instaces buffer, 256 KB feature cache, 8 KB edge buffer, 8 FP32 adder and 8 FP32 mult for each rank-AU

5 EVALUATION

5.1 Experimental Setup

Simulation. The basic configuration of MetaNMP is summarised in Table 2. We integrate both Ramulator [32] and ZSim [39], a cycle-level accurate simulator, to evaluate the performance of MetaNMP. We modify the trace-driven version of ZSim to simulate the host for offloading the NMP-inst, reading edge and feature data, and broadcasting data to the NMP side. In Ramulator, we add the DIMM-MetaNMP module and the rank-AU module to simulate the metapath instance generation and feature aggregation, respectively. We also modify Ramulator to implement our broadcast mechanism. To generate traces required for ZSim, we simulate HGNNs at the behavioral level, which considers that feature and edge data may be mapped randomly to different ranks by OS. We estimate the cache area, energy consumption, and access latency using Cacti [21]. The area, energy consumption, and latency of the special hardware units we integrate are evaluated with Synopsys Design Compiler under a 40nm technology library. Cacit-3DD [7] is used to evaluate the energy consumption of DRAM devices, and Cacti-IO [28] is used to evaluate the energy consumption of off-chip I/O at the DIMM-level.

Methodology. We compare MetaNMP with five state-of-the-art designs on typical platforms. (1) Baseline: Intel Xeon Gold 5117 CPU, where we improve the baseline CPU performance using our software optimization; (2) The NVIDIA Tesla V100 GPU; (3) AWB-GCN [16]: A graph convolutional neural networks accelerator with a runtime auto-tuning technique to solve the load imbalance problem among PEs. We convert the aggregation of HGNNs into matrix computations on AWB-GCN; (4) HyGCN [49]: a hybrid accelerator with customized hardware for aggregation and combination of graph convolutional neural networks. We decompose the complex metapath-based aggregation into simple vertex aggregation on HyGCN; (5) RecNMP [30]: a DIMM-based near-memory processing accelerator for accelerating embedding and tensor operations in deep learning. To simulate HGNNs on RecNMP, we offload the aggregation to the NMP side, and aggregation instructions are sent

Table 3: Datasets

	#Vertex	#Edge	Metapath
DBLP (DP) [13]	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) [12]	Movie(M): 4278 Director(D): 2081 Actor(A): 5257	M-D: 4278 M-A: 12828	MDM MAM DMD DMAMD AMA AMDMA
LastFM (LF) [12]	User(U): 1892 Artist(A): 17632 Tag(T): 1088	U-U: 12717 U-A: 92834 A-T: 23253	UAU UATAU AUA ATA
OGB- MAG (OM) [24]	Author(A): 1134649 Paper(P): 36389 Institution(I): 8740 Field(F): 59965	A-I: 1043998 A-P: 7145660 P-P: 5416271 P-F: 7505078	APA APFPA
OAG (OG) [40]	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

directly by the host. Since metapath instance generation on AWB-GCN, HyGCN, and RecNMP is not implemented, we use MetaNMP to generate instances and then add this time to them.

Workloads. The datasets are shown in Table 3. We update the graph at a batch granularity, where each batch contains 10% of the graph changes. After each graph update, one inference is performed. This is similar to previous dynamic scenarios [5]. We use three representative HGNNs: (1) **MAGNN** [12] aggregates the structural information within the metapath instance and aggregates the semantic information among metapaths. (2) **HAN** [45] only aggregates metapath-based neighbors (i.e., the end vertex of the metapath instance) for structural information and performs semantic aggregation based on the results of structural aggregation to obtain the final vertex embedding. (3) **SHGNN** [48] constructs tree structures. It aggregates structural information within the tree structure and performs semantic information aggregation among different trees.

5.2 Memory Consumption Efficiency

The pre-processing of metapath instance matching requires storing all instances, which causes significant memory consumption. Especially for large graphs, such as OGB-MAG, the memory consumption required in the model is high due to the hundreds of millions of metapath instances to store. Our approach generates metapath instances on the fly. Experimental results show that MetaNMP reduces memory consumption by 51.9% on average, as shown in Table 4. MetaNMP does not need to store metapath instances, and is very friendly to HGNNs applied on real-world large graphs.

5.3 Overall Performance

Figure 12 shows the performance results of MetaNMP against CPU, GPU, AWB-GCN [16], HyGCN [49], and RecNMP [30].

Table 4: Memory consumption reduction ratio of MetaNMP

	MAGNN	HAN	SHGNN
DP-APVPA	44%	39%	46%
DP-APTPA	41%	36%	45%
IB-AMDMA	13%	12%	15%
LF-AUA	29%	26%	33%
LF-ATA	33%	30%	36%
LF-UAU	25%	26%	28%
LF-UATAU	84%	74%	96%
OM-APA	76%	65%	79%
OM-APFPA	88%	76%	89%
OG-APA	65%	64%	69%
OG-APFPA	79%	70%	82%

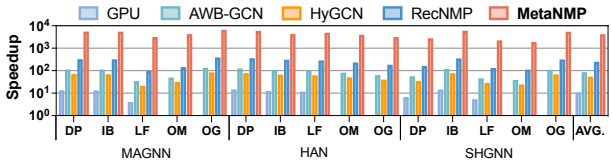

Figure 12: Speedup of MetaNMP compared with GPU, AWB-GCN, HyGCN, and RecNMP. All results are normalized to CPU. OM and OG fail in running on GPU due to out of memory.

1) **MetaNMP vs. CPU and GPU.** CPU and GPU are the platforms commonly used to accelerate HGNNs. On average, MetaNMP outperforms CPU by 4225.51×. Compared to GPU, MetaNMP is on average 415.18× faster. This is because MetaNMP is customized with specialized hardware units to efficiently generate and sense metapath instances. It also utilizes near-memory processing to overcome the memory bottleneck of HGNN. As we can see, MetaNMP has a higher performance improvement in MAGNN compared to HAN and SHGNN. Because MAGNN has a large number of redundant computations among metapath instances, our relevant metapath instance awareness can eliminate these redundant computations.

2) **MetaNMP vs. AWB-GCN and HyGCN.** AWB-GCN and HyGCN are accelerators of *graph neural networks* (GNNs) on homogeneous graphs. MetaNMP is on average 48.96× and 78.34× faster than AWB-GCN and HyGCN, respectively. AWB-GCN accelerates traditional GNNs with sparse matrix multiplication, we convert complex metapath-based aggregation into matrix multiplication. HyGCN uses a hybrid engine to fuse the memory-bound aggregation and the compute-bound combination efficiently. However, in HGNNs, the dominant computations are memory-bound, and the only compute-bound feature projection phase is usually performed first. It is difficult to adapt HyGCN's hybrid engine for fusion. These two accelerators are not designed for HGNNs and are difficult to use for accelerating. Based on the characteristics of HGNNs, MetaNMP supports fast metapath instance generation and efficient feature aggregation, and further overcomes memory bottlenecks by leveraging near-memory processing.

3) **MetaNMP vs. RecNMP.** We also compare MetaNMP with the state-of-the-art RecNMP. RecNMP integrates customized hardware for each rank to exploit rank-level bandwidth for near-memory processing. MetaNMP is 17.23× faster than RecNMP. The reason

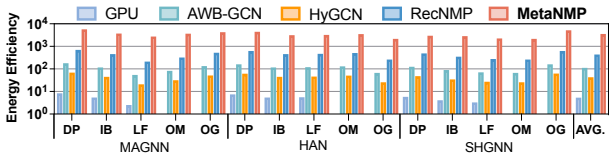

Figure 13: Energy efficiency of MetaNMP compared with GPU, AWB-GCN, HyGCN, and RecNMP. All results are normalized to CPU.

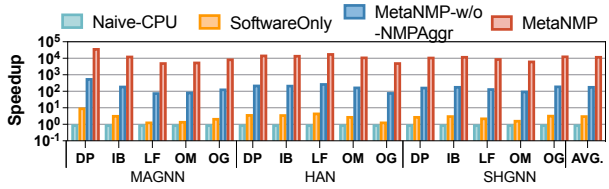

Figure 14: Performance of MetaNMP in different configurations. SoftwareOnly implements our approach through software only without NMP hardware. MetaNMP-W/o-NMPAggr generates metapath instances through NMP hardware without performing aggregation in rank-level.

behind this is mainly from two aspects. One is our regular data broadcast approach which alleviates the communication among DIMMs. The other is our metapath instance-driven aggregation, which is directly guided by the DIMM-MetaNMP module to perform without additional help from the host.

4) **Energy Savings.** Figure 13 shows the energy results of MetaNMP against CPU, GPU, AWB-GCN [16], HyGCN [49], and RecNMP [30]. MetaNMP consumes on average 3563.25× less energy than CPU and 690.13× less energy than GPU. MetaNMP outperforms AWB-GCN on average with 31.98× less energy consumption. Also, MetaNMP is superior to HyGCN with less energy consumption by 83.15× on average. This is because near-memory processing significantly reduces data movement. MetaNMP has on average 8.36× less energy consumption than RecNMP. The reasons are twofold. First, the broadcast mechanism reduces communication among DIMMs. Second, metapath instance-driven aggregation is directly guided by the DIMM-MetaNMP module to perform without the help of the host.

5.4 Software-only Performance Improvements

Figure 14 compares the performance of MetaNMP under different configurations. SoftwareOnly is a software-only implementation of our approach, where the CPU generates metapath instances on the fly, senses the metapath instances to avoid redundant computations, and guides feature aggregation of metapath instances. The SoftwareOnly approach is only 3.54× faster than the naive CPU, due to the high overhead caused by the data-dependent instructions and monitoring the relationships among metapath instances. MetaNMP-w/o-NMPAggr generates and senses metapath instances with near-memory processing, and the CPU performs the aggregation computation. As seen, MetaNMP-w/o-NMPAggr is 60.23× faster than SoftwareOnly. Finally, MetaNMP shows a higher performance improvement of 3963.12× than SoftwareOnly because it exploits rank-level bandwidth for aggregation computation.

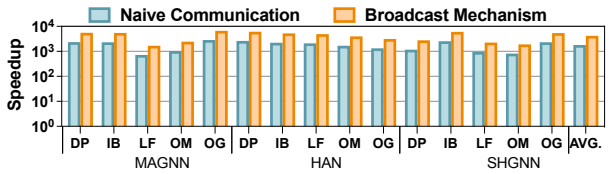

Figure 15: Performance comparison of MetaNMP using naive communication and broadcast mechanism

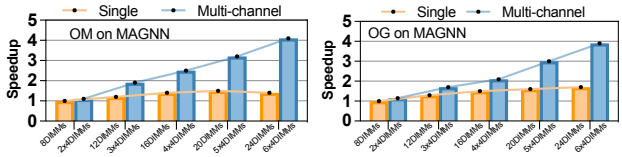

Figure 16: Scalability of MetaNMP on large heterogeneous graphs using single channel and multi-channel as the number of DIMMs increases

5.5 Effectiveness of Broadcast Mechanism

To demonstrate the effectiveness of our proposed broadcast mechanism, we compare it with the naive approach. In the naive approach, each DIMM is autonomously responsible for its own metapath instance generation and vertex aggregation. They directly request the data with the help of the host. Figure 15 shows the performance results of comparing MetaNMP with the broadcast mechanism and the naive communication approach. We can see that MetaNMP using the broadcast mechanism is 2.35× faster than the naive approach. It is easy for the naive approach to become the performance bottleneck because different DIMMs repeatedly request the same edge data and vertex feature data. On the contrary, we broadcast data among DIMMs. Thus, the data can be shared among them simultaneously to reduce the overhead caused by communication.

5.6 Scalability

In this subsection, we evaluate performance scalability of MetaNMP with varying the number of DIMMs and ranks.

#DIMMs. Figure 16 shows the performance scalability of MetaNMP on large heterogeneous graphs using single channel and multi-channel as the number of DIMMs increases. We can see that adding more DIMMs to a single channel does not increase performance significantly, as the broadcast mechanism becomes the bottleneck. However, the existing CPU architecture is usually implemented in a multi-channel manner. In such multi-channel systems, the broadcast mechanism is only used when multiple DIMMs within a channel require the same data. CPU broadcasts data to each channel one by one. Thus, broadcasting data to other channels is unnecessary when only a single channel requires the data. The performance of MetaNMP increases with the number of DIMMs under a multi-channel setting, which shows that the broadcast mechanism does not become the bottleneck in this case.

#Ranks. Figure 17 also shows the performance of MetaNMP varies with the number of ranks. Although the rank-level bandwidth increases with the number of ranks, the DIMM-level bandwidth does not increase linearly due to the shared data cables between ranks. However, MetaNMP still increases the performance with the number of ranks, e.g., the performance improvement is 1.96× for 4

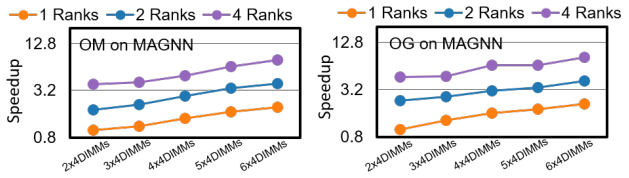

Figure 17: Performance of MetaNMP with different number of ranks

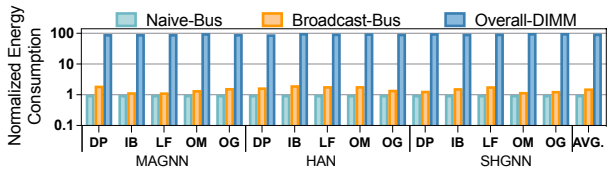

Figure 18: Energy consumption of the bus on naive communication, the bus on broadcast communication, and the overall NMP-enabled DIMM system

ranks compared to 2 ranks. This is because we exploit the rank-level bandwidth with near-memory processing. We integrate the aggregation unit in each rank. Thus, as the number of ranks increases, the higher aggregation bandwidth can be given to MetaNMP. This shows that MetaNMP scales better with the number of ranks.

5.7 Overhead

Inter-DIMM Broadcast. Inter-DIMM broadcast communication has no additional hardware overhead and requires slight modifications to the controller's Finite State Machines and buffer chip in DIMM. The inter-DIMM broadcast causes only a negligible increase in energy consumption on the bus, as shown in Figure 18. The energy consumption of broadcast communication on the bus is on average 1.61× higher than the naive communication, but only accounts for 1.3% of the overall DIMM memory system. This is because DIMMs are NMP-enabled and support parallel operations on multiple DIMMs, so the increase in broadcast communication energy consumption is trivial on such an NMP-based DIMM system.

Overall Area and Power Analysis. Table 5 shows the area and power consumption of MetaNMP. Under 40nm technology, MetaNMP has a very small area overhead of 0.8026mm^2, which is much smaller than the 10mm^2 area of a typical DRAM chip [8]. The power of MetaNMP is 129.84mW, which is also much smaller than the entire Load-Reduced DIMM (LRDIMM) power of 10W [8].

6 RELATED WORK

Heterogeneous Graph Neural Networks. PyG [11] and DGL [44] are frameworks supporting graph neural networks on heterogeneous graphs. Yan et al. [50] investigate the characteristics of HGNN on GPU. HGNNs suffer from many problems, yet receive little attention from computer architecture. Our work bridges this gap.

Homogeneous Graph Neural Networks. There are a number of efforts to accelerate graph neural networks on homogeneous graphs. At the software level, NeuGraph [36] proposes a SAGA programming model and supports multiple GPUs. Dorylus [42] presents a computation separation technique and implements a distributed GNN system that can scale to compute on large graphs.

Table 5: The area and power of MetaNMP

	Rank-AU	DIMM-MetaNMP	Total	Typical DRAM chip
Area (mm^2)	0.7045	0.0981	0.8026	100mm^2
Power (mW)	113.34	16.5	129.84	10W

GNNAdvisor [46] proposes a group-based workload management to improve the performance on GPU.

At the hardware level, HyGCN [49] proposes a hybrid architecture for GCN. AWB-GCN [16] presents three autotuning techniques to balance the workload. GCNAX [35] proposes a flexible dataflow on a sparse dense matrix multiplication to accelerate GCN. Auten [3] proposes a modular architecture for the three phases of graph traversal, combination, and aggregation of GNNs. Also, PIM-GCN [4], DARe [2], TARe [18], and ReFlip [25] are based on the ReRAM architecture to accelerate GNNs. These efforts are based on traditional graph neural networks, and HGNNs differ significantly. HGNNs aggregate the structural and semantic information based on metapath, which is more complex than traditional GNNs that aggregate neighbor information directly.

Near-Memory Processing. There are a lot of works to exploit near-memory processing. Many previous efforts were based on 3D stacked DRAM technology [1, 9, 10, 14, 15, 22, 31, 38, 43], which has a limited capacity (16-32GB) and is unfriendly to HGNNs on large graphs. Chameleon [37] proposes DIMM-based near-memory processing for large memory systems. RecNMP [30] and Space [29] design DIMM-based near-memory processing accelarators for recommended systems. These near-memory processing accelerators are hardly applied to accelerate HGNNs. In this work, MetaNMP solves the memory consumption of storing metapath instances, sensing the relationship among metapath instances, and overcoming memory bottlenecks based on the characteristics of HGNNs.

7 CONCLUSION

In this paper, we propose MetaNMP, a NMP-based HGNNs accelerator. MetaNMP is featured with the following key designs. First, we propose a cartesian-like product approach to generate metapath instances on the fly for avoiding significant memory consumption. Second, we design a data flow to exploit the shareable aggregation computation among metapath instances. Third, we implement MetaNMP with DIMM-based near-memory processing, and use broadcast mechanism to reduce the inter-DIMM communication. Experimental results show that MetaNMP obtains memory space reduction of 51.9% on average, while achieving better performance improvement compared to existing HGNNs systems.

ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers for their insightful comments. This work is supported by the National Key Research and Development Program of China (Grant No. 2022YFB-4501403), the National Natural Science Foundation of China (Grant No. 61832006, 61825202, and 61929103), and Huawei Technologies Co., Ltd (Grant No. YBN2021035018).

REFERENCES

[1] Junwhan Ahn, Sungpack Hong, Sungjoo Yoo, Onur Mutlu, and Kiyoung Choi. 2015. A Scalable Processing-In-Memory Accelerator for Parallel Graph Processing. In *Proceedings of the Annual International Symposium on Computer Architecture (ISCA)*. 105–117.

[2] Aqeeb Iqbal Arka, Biresh Kumar Joardar, Janardhan Rao Doppa, Partha Pratim Pande, and Krishnendu Chakrabarty. 2021. DARe: DropLayer-Aware Manycore ReRAM Architecture for Training Graph Neural Networks. In *Proceedings of the IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. 1–9.

[3] Adam Auten, Matthew Tomei, and Rakesh Kumar. 2020. Hardware Acceleration of Graph Neural Networks. In *Proceedings of the Design Automation Conference (DAC)*. 1–6.

[4] Nagadastagiri Challapalle, Karthik Swaminathan, Nandhini Chandramoorthy, and Vijaykrishnan Narayanan. 2021. Crossbar based Processing in Memory Accelerator Architecture for Graph Convolutional Networks. In *Proceedings of the IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. 1–9.

[5] Dan Chen, Chuangyi Gui, Yi Zhang, Hai Jin, Long Zheng, Yu Huang, and Xiaofei Liao. 2022. GraphFly: Efficient Asynchronous Streaming Graphs Processing via Dependency-Flow. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*. 632–645.

[6] Dan Chen, Hai Jin, Long Zheng, Yu Huang, Pengcheng Yao, Chuangyi Gui, Qinggang Wang, Haifeng Liu, Haiheng He, Xiaofei Liao, and Ran Zheng. 2022. A General Offloading Approach for Near-DRAM Processing-In-Memory Architectures. In *Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 246–257.

[7] Ke Chen, Sheng Li, Naveen Muralimanohar, Jung Ho Ahn, Jay B. Brockman, and Norman P. Jouppi. 2012. CACTI-3DD: Architecture-level Modeling for 3D Die-stacked DRAM Main Memory. In *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 33–38.

[8] Guohao Dai, Zhenhua Zhu, Tianyu Fu, Chiyue Wei, Bangyan Wang, Xiangyu Li, Yuan Xie, Huazhong Yang, and Yu Wang. 2022. DIMMining: Pruning-Efficient and Parallel Graph Mining on Near-Memory-Computing. In *Proceedings of the Annual International Symposium on Computer Architecture (ISCA)*. 130–145.

[9] Mario Drumond, Alexandros Daglis, Nooshin Sadat Mirzadeh, Dmitrii Ustiugov, Javier Picorel, Babak Falsafi, Boris Grot, and Dionisios N. Pnevmatikatos. 2017. The Mondrian Data Engine. In *Proceedings of the Annual International Symposium on Computer Architecture (ISCA)*. 639–651.

[10] Amin Farmahini Farahani, Jung Ho Ahn, Katherine Morrow, and Nam Sung Kim. 2015. NDA: Near-DRAM Acceleration Architecture Leveraging Commodity DRAM Devices and Standard Memory Modules. In *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*. 283–295.

[11] Matthias Fey and Jan Eric Lenssen. 2019. Fast Graph Representation Learning with PyTorch Geometric. *CoRR* abs/1903.02428 (2019).

[12] Xinyu Fu, Jiani Zhang, Ziqiao Meng, and Irwin King. 2020. MAGNN: Metapath Aggregated Graph Neural Network for Heterogeneous Graph Embedding. In *Proceedings of the World Wide Web Conference (WWW)*. 2331–2341.

[13] Jing Gao, Feng Liang, Wei Fan, Yizhou Sun, and Jiawei Han. 2009. Graph-based Consensus Maximization among Multiple Supervised and Unsupervised Models. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*. 585–593.

[14] Mingyu Gao, Grant Ayers, and Christos Kozyrakis. 2015. Practical Near-Data Processing for In-Memory Analytics Frameworks. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques (PACT)*. 113–124.

[15] Mingyu Gao, Jing Pu, Xuan Yang, Mark Horowitz, and Christos Kozyrakis. 2017. TETRIS: Scalable and Efficient Neural Network Acceleration with 3D Memory. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. 751–764.

[16] Tong Geng, Ang Li, Runbin Shi, Chunshu Wu, Tianqi Wang, Yanfei Li, Pouya Haghi, Antonino Tumeo, Shuai Che, Steven K. Reinhardt, and Martin C. Herbordt. 2020. AWB-GCN: A Graph Convolutional Network Accelerator with Runtime Workload Rebalancing. In *Proceedings of the Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 922–936.

[17] Xue Geng, Hanwang Zhang, Jingwen Bian, and Tat-Seng Chua. 2015. Learning Image and User Features for Recommendation in Social Networks. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. 4274–4282.

[18] Yintao He, Ying Wang, Cheng Liu, Huawei Li, and Xiaowei Li. 2021. TARe: Task-Adaptive in-situ ReRAM Computing for Graph Learning. In *Proceedings of the ACM/IEEE Design Automation Conference (DAC)*. 577–582.

[19] Changwan Hong, Aravind Sukumaran-Rajam, Israt Nisa, Kunal Singh, and Ponnuswamy Sadayappan. 2019. Adaptive Sparse Tiling for Sparse Matrix Multiplication. In *Proceedings of the ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP)*. 300–314.

[20] Shifu Hou, Yanfang Ye, Yangqiu Song, and Melih Abdulhayoglu. 2017. HinDroid: An Intelligent Android Malware Detection System Based on Structured Heterogeneous Information Network. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. 1507–1515.

[21] HP laboratories. 2009. CACTI 6.0: A tool to model large caches. Available: https://www.hpl.hp.com/techreports/2009/HPL-2009-85.pdf.

[22] Kevin Hsieh, Eiman Ebrahimi, Gwangsun Kim, Niladrish Chatterjee, Mike O'Connor, Nandita Vijaykumar, Onur Mutlu, and Stephen W. Keckler. 2016. Transparent Offloading and Mapping (TOM): Enabling Programmer-Transparent Near-Data Processing in GPU Systems. In *Proceedings of the ACM/IEEE Annual International Symposium on Computer Architecture (ISCA)*. 204–216.

[23] Binbin Hu, Zhiqiang Zhang, Chuan Shi, Jun Zhou, Xiaolong Li, and Yuan Qi. 2019. Cash-Out User Detection Based on Attributed Heterogeneous Information Network with a Hierarchical Attention Mechanism. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*. 946–953.

[24] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. 2020. Open Graph Benchmark: Datasets for Machine Learning on Graphs. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*. 4869–4880.

[25] Yu Huang, Long Zheng, Pengcheng Yao, Qinggang Wang, Xiaofei Liao, Hai Jin, and Jingling Xue. 2022. Accelerating Graph Convolutional Networks Using Crossbar-based Processing-In-Memory Architectures. In *Proceedings of the IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 1029–1042.

[26] Yu Huang, Long Zheng, Pengcheng Yao, Qinggang Wang, Haifeng Liu, Xiaofei Liao, Hai Jin, and Jingling Xue. 2022. ReaDy: A ReRAM-Based Processing-In-Memory Accelerator for Dynamic Graph Convolutional Networks. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* (2022), 3567–3578. https://doi.org/10.1109/TCAD.2022.3199152

[27] Hai Jin, Dan Chen, Long Zheng, Yu Huang, Pengcheng Yao, Jin Zhao, Xiaofei Liao, and Wenbin Jiang. 2023. Accelerating Graph Convolutional Networks Through a PIM-Accelerated Approach. *IEEE Trans. Comput.* (2023), 1–12. https://doi.org/10.1109/TC.2023.3257514

[28] Norman P. Jouppi, Andrew B. Kahng, Naveen Muralimanohar, and Vaishnav Srinivas. 2012. Cacti-io: Cacti with Off-chip Power-Area-Timing Models. In *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*. 294–301.

[29] Hongju Kal, Seokmin Lee, Gun Ko, and Won Woo Ro. 2021. SPACE: Locality-Aware Processing in Heterogeneous Memory for Personalized Recommendations. In *Proceedings of the ACM/IEEE Annual International Symposium on Computer Architecture (ISCA)*. 679–691.

[30] Liu Ke, Udit Gupta, Benjamin Youngjae Cho, David Brooks, Vikas Chandra, Utku Diril, Amin Firoozshahian, Kim M. Hazelwood, Bill Jia, Hsien-Hsin S. Lee, Meng Li, Bert Maher, Dheevatsa Mudigere, Maxim Naumov, Martin Schatz, Mikhail Smelyanskiy, Xiaodong Wang, Brandon Reagen, Carole-Jean Wu, Mark Hempstead, and Xuan Zhang. 2020. RecNMP: Accelerating Personalized Recommendation with Near-Memory Processing. In *Proceedings of the ACM/IEEE Annual International Symposium on Computer Architecture (ISCA)*. 790–803.

[31] Duckhwan Kim, Jaeha Kung, Sek M. Chai, Sudhakar Yalamanchili, and Saibal Mukhopadhyay. 2016. Neurocube: A Programmable Digital Neuromorphic Architecture with High-Density 3D Memory. In *Proceedings of the ACM/IEEE Annual International Symposium on Computer Architecture (ISCA)*. 380–392.

[32] Yoongu Kim, Weikun Yang, and Onur Mutlu. 2015. Ramulator: A Fast and Extensible DRAM Simulator. *IEEE Computer Architecture Letters* (2015), 45–49. https://doi.org/10.1109/LCA.2015.2414456

[33] Miryeong Kwon, Donghyun Gouk, Sangwon Lee, and Myoungsoo Jung. 2022. Hardware/Software Co-Programmable Framework for Computational SSDs to Accelerate Deep Learning Service on Large-Scale Graphs. In *Proceedings of the USENIX Conference on File and Storage Technologies (FAST)*. 147–164.

[34] Yunjae Lee, Jinha Chung, and Minsoo Rhu. 2022. SmartSAGE: Training Large-scale Graph Neural Networks using In-Storage Processing Architectures. In *Proceedings of the Annual International Symposium on Computer Architecture (ISCA)*. 932–945.

[35] Jiajun Li, Ahmed Louri, Avinash Karanth, and Razvan C. Bunescu. 2021. GCNAX: A Flexible and Energy-efficient Accelerator for Graph Convolutional Neural Networks. In *Proceedings of the IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. 775–788.

[36] Lingxiao Ma, Zhi Yang, Youshan Miao, Jilong Xue, Ming Wu, Lidong Zhou, and Yafei Dai. 2019. NeuGraph: Parallel Deep Neural Network Computation on Large Graphs. In *Proceedings of USENIX Annual Technical Conference (ATC)*. 443–458.

[37] Hadi Asghari Moghaddam, Young Hoon Son, Jung Ho Ahn, and Nam Sung Kim. 2016. Chameleon: Versatile and Practical Near-DRAM Acceleration Architecture for Large Memory Systems. In *Proceedings of the Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 50:1–50:13.

[38] Lifeng Nai, Ramyad Hadidi, Jaewoong Sim, Hyojong Kim, Pranith Kumar, and Hyesoon Kim. 2017. GraphPIM: Enabling Instruction-Level PIM Offloading in Graph Computing Frameworks. In *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*. 457–468.

[39] Daniel Sanchez and Christos Kozyrakis. 2013. ZSim: Fast and Accurate Microarchitectural Simulation of Thousand-Core Systems. *ACM SIGARCH Computer Architecture News* (2013), 475–486. https://doi.org/10.1145/2508148.2485963

[40] Arnab Sinha, Zhihong Shen, Yang Song, Hao Ma, Darrin Eide, Bo-June Paul Hsu, and Kuansan Wang. 2015. An Overview of Microsoft Academic Service (MAS)

and Applications. In *Proceedings of the World Wide Web Conference (WWW)*. 243–246.

[41] Weiyi Sun, Zhaoshi Li, Shouyi Yin, Shaojun Wei, and Leibo Liu. 2021. ABC-DIMM: Alleviating the Bottleneck of Communication in DIMM-based Near-Memory Processing with Inter-DIMM Broadcast. In *Proceedings of the ACM/IEEE Annual International Symposium on Computer Architecture (ISCA)*. 237–250.

[42] John Thorpe, Yifan Qiao, Jonathan Eyolfson, Shen Teng, Guanzhou Hu, Zhihao Jia, Jinliang Wei, Keval Vora, Ravi Netravali, Miryung Kim, and Guoqing Harry Xu. 2021. Dorylus: Affordable, Scalable, and Accurate GNN Training with Distributed CPU Servers and Serverless Threads. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. 495–514.

[43] Po-An Tsai, Changping Chen, and Daniel Sánchez. 2018. Adaptive Scheduling for Systems with Asymmetric Memory Hierarchies. In *Proceedings of the IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 641–654.

[44] Minjie Wang, Lingfan Yu, Da Zheng, Quan Gan, Yu Gai, Zihao Ye, Mufei Li, Jinjing Zhou, Qi Huang, Chao Ma, Ziyue Huang, Qipeng Guo, Hao Zhang, Haibin Lin, Junbo Zhao, Jinyang Li, Alexander J. Smola, and Zheng Zhang. 2019. Deep Graph Library: Towards Efficient and Scalable Deep Learning on Graphs. *CoRR* abs/1909.01315 (2019).

[45] Xiao Wang, Houye Ji, Chuan Shi, Bai Wang, Yanfang Ye, Peng Cui, and Philip S. Yu. 2019. Heterogeneous Graph Attention Network. In *Proceedings of the World Wide Web Conference (WWW)*. 2022–2032.

[46] Yuke Wang, Boyuan Feng, Gushu Li, Shuangchen Li, Lei Deng, Yuan Xie, and Yufei Ding. 2021. GNNAdvisor: An Adaptive and Efficient Runtime System for GNN Acceleration on GPUs. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. 515–531.

[47] Samuel Williams, Andrew Waterman, and David Patterson. 2009. Roofline: An Insightful Visual Performance Model for Multicore Architectures. *Commun. ACM* (2009), 65–76. https://doi.org/10.1145/1498765.1498785

[48] Wentao Xu, Yingce Xia, Weiqing Liu, Jiang Bian, Jian Yin, and Tie-Yan Liu. 2021. SHGNN: Structure-Aware Heterogeneous Graph Neural Network. *CoRR* abs/2112.06244 (2021).

[49] Mingyu Yan, Lei Deng, Xing Hu, Ling Liang, Yujing Feng, Xiaochun Ye, Zhimin Zhang, Dongrui Fan, and Yuan Xie. 2020. HyGCN: A GCN Accelerator with Hybrid Architecture. In *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*. 15–29.

[50] Mingyu Yan, Mo Zou, Xiaocheng Yang, Wenming Li, Xiaochun Ye, Dongrui Fan, and Yuan Xie. 2022. Characterizing and Understanding HGNNs on GPUs. *IEEE Computer Architecture Letters* (2022), 69–72. https://doi.org/10.1109/LCA.2022.3198281

[51] Michihiro Yasunaga, Jungo Kasai, Rui Zhang, Alexander R. Fabbri, Irene Li, Dan Friedman, and Dragomir R. Radev. 2019. ScisummNet: A Large Annotated Corpus and Content-Impact Models for Scientific Paper Summarization with Citation Networks. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*. 7386–7393.

[52] Huan Zhao, Quanming Yao, Jianda Li, Yangqiu Song, and Dik Lun Lee. 2017. Meta-Graph Based Recommendation Fusion over Heterogeneous Information Networks. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. 635–644.

[53] Long Zheng, Haifeng Liu, Yu Huang, Dan Chen, Chaoqiang Liu, Haiheng He, Xiaofei Liao, Hai Jin, and Jingling Xue. 2022. A Flexible Yet Efficient DNN Pruning Approach for Crossbar-Based Processing-in-Memory Architectures. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* (2022), 3745–3756. https://doi.org/10.1109/TCAD.2022.3197510

[54] Xiaoping Zhou, Xun Liang, Haiyan Zhang, and Yuefeng Ma. 2016. Cross-Platform Identification of Anonymous Identical Users in Multiple Social Media Networks. *IEEE Trans. Knowl. Data Eng.* (2016), 411–424. https://doi.org/10.1109/TKDE.2015.2485222