Lift: Exploiting Hybrid Stacked Memory for Energy-Efficient Processing of Graph Convolutional Networks

Jiaxian Chen, Zhaoyu Zhong, Kaoyi Sun, Chenlin Ma, Rui Mao, Yi Wang
College of Computer Science and Software Engineering, Shenzhen University, Shenzhen, China

***Abstract*—Graph Convolutional Networks (GCNs) are powerful learning approaches for graph-structured data. GCNs are both computing- and memory-intensive. The emerging 3D-stacked computation-in-memory (CIM) architecture provides a promising solution to process GCNs efficiently. The CIM architecture can provide near-data computing, thereby reducing data movement between computing logic and memory. However, previous works do not fully exploit the CIM architecture in both dataflow and mapping, leading to significant energy consumption.**

This paper presents *Lift*, an energy-efficient GCN accelerator based on 3D CIM architecture using software and hardware co-design. At the hardware level, Lift introduces a hybrid architecture to process vertices with different characteristics. Lift adopts near-bank processing units with a push-based dataflow to process vertices with strong re-usability. A dedicated unit is introduced to reduce massive data movement caused by high-degree vertices. At the software level, Lift adopts a hybrid mapping to further exploit data locality and fully utilize the hybrid computing resources. The experimental results show that the proposed scheme can significantly reduce data movement and energy consumption compared with representative schemes.

***Index Terms*—3D-Stacked Memory, Computation-in-Memory, Graph Convolutional Networks, Accelerator.**

I. Introduction

Graph convolutional networks (GCNs) are popular learning approaches for graph-structured data [1]. GCNs are widely applied to social networks, recommendation systems, and other specific domains [2]. GCNs mainly consist of two time-consuming phases: (1) *combination* and (2) *aggregation*. These two phases have different computation and memory access characteristics. The combination phase typically uses a multi-layer perceptron (MLP), while the aggregation phase acts like graph processing, where each vertex aggregates features from its neighboring vertices. In the aggregation phase, significant data movement is incurred by the sparse and uncertain topology of the input graph, which poses a severe challenge to the energy-efficient processing of GCNs.

Many domain-specific architectures have been proposed to address the challenge for GCNs [3–5]. Some previous works focus on reducing redundant memory accesses and computation [6–8]. Although these works can effectively improve the performance of GCNs, they still suffer from enormous data movement due to the sparse input graph. Some works rely on dedicated buffers to alleviate irregular data access [4, 9]. However, the large dimension of the vertex features seriously degrades the efficiency of the on-chip buffers. The above works only partially solve the problem by reducing random memory accesses. The data movement issue still exists, which leads to significant energy consumption.

The computation-in-memory architecture provides a promising solution to reduce data movement. The advanced 3D memory stacks multiple DRAM dies on a base die, offering considerable capacity and bandwidth. With the emerging integrated circuit technology, computing units can be placed in memory to take advantage of the huge DRAM capacity for data reuse. Therefore, data movement between memory and computing units can be significantly reduced. The recent work GCIM [10] adopts the 3D CIM architecture with near-bank CIM units to accelerate GCNs. GCIM does not fully exploit the 3D CIM architecture, leading to significant cross-bank data movement and energy consumption. We argue that the data movement issue mainly comes from unoptimized mapping and dataflow.

To address these challenges, this paper present *Lift*, an energy-efficient GCN accelerator based on 3D CIM architectures. The objective is to fully exploit the 3D CIM architecture and reduce energy consumption caused by data movement. As a hardware and software co-design solution, Lift optimizes the 3D CIM architecture and the processing of GCNs. At the hardware level, Lift proposes a near-bank processing unit with push-based dataflow for efficient data reuse. A dedicated processing unit is also introduced to reduce massive data movement incurred by high-degree vertices. At the software level, Lift presents a hybrid mapping to exploit data locality and fully utilize the hybrid computing resources. Lift is compared with the baseline scheme GCIM in terms of energy, latency, and data movement. The experimental results show that the proposed scheme can reduce data movement by 86% on average and achieve 6.05$\times$ energy reduction in comparison with baseline schemes.

The main contributions of this work can be summarized as follows:

- A near-bank processing CIM architecture is proposed to leverage data locality and alleviate cross-bank data fetching caused by high-degree vertices.
- A hybrid mapping strategy is proposed to reduce data movement and facilitate the proposed architecture.
- As a proof of concept, we compare the proposed approach with representative schemes using a set of GCN workloads and graph datasets.

The rest of this paper is organized as follows. Section II provides an overview of the background and discusses the motivation of this paper. Section III presents the proposed technique in detail. Section IV shows the experimental results with discussion. Finally, in Section V, we conclude this paper and discuss the future work.

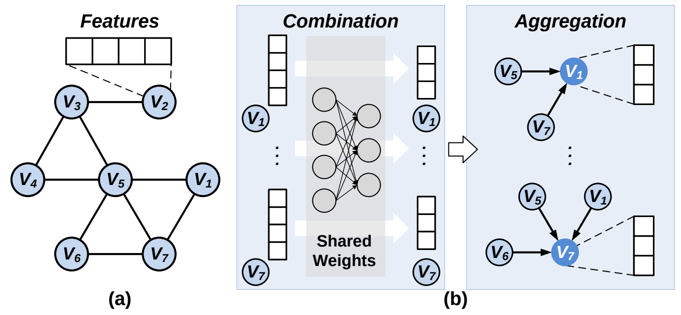

Fig. 1: (a) Input graph. (b) Inference procedure of a typical GCN model.

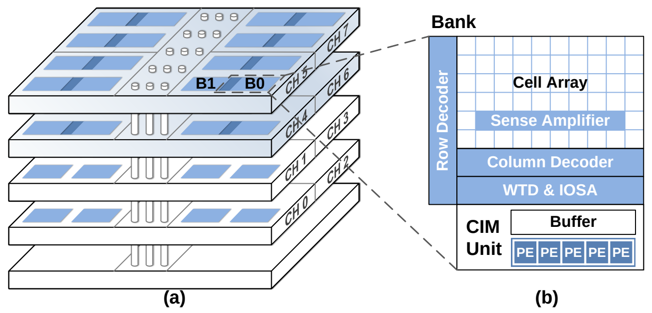

Fig. 2: (a) A typical HBM-based 3D CIM architecture. (b) A bank with a CIM unit.

II. Background and Motivation

A. Graph Convolutional Networks

Figure 1 illustrates the inference procedure of a typical GCN model with one graph convolutional layer. The input of a GCN model is a graph $G(V, E)$ with feature properties (Figure 1(a)), where V and E denote vertices and edges, respectively. Within the input graph, the features of each vertex are represented by a feature vector. As shown in Figure 1(b), a graph convolutional layer typically consists of two phases: combination and aggregation. The features of each vertex are updated continuously with the execution of these phases. In the combination phase, the feature vector of each vertex is transformed into a new one via *Combine* functions. In the aggregation phase, each vertex gathers all features of its neighboring vertices and then generates a new feature vector using *Aggregate* functions (e.g., Mean function). For example, in Figure 1(b), V_1 gathers features of its neighbors V_5 and V_7, while V_7 gathers features from its neighbors V_1, V_5, and V_6. After applying multiple graph convolutional layers, the output features can extract the high-level structural information from the input graph. The k-th layer can be abstracted as:

$$X^{k+1} = \sigma(\hat{A}X^kW^k) \tag{1}$$

where $\hat{A}$ is a sparse matrix derived from E. X^k and W^k are matrices. They represent features of vertices and weights of the MLP model in k-th layer, respectively. σ denotes non-linear activation functions. Since each layer is abstracted into matrix chain multiplication, the execution order between the combination and aggregation phases can be changed. Previous works [3, 9] show that the combination first approach $(\hat{A}(X^kW^k))$ can reuse the sparse-dense matrix multiplication (SpMM) kernel for these two phases and reduce arithmetic computation. Therefore, the proposed Lift adopts the combination first approach.

B. 3D Computing-in-Memory Architecture

Figure 2 shows the 3D CIM architecture based on high bandwidth memory (HBM) [11]. With the emerging 3D-stacking technology, multiple DRAM dies stack on the top of a base die, providing significant memory bandwidth. DRAM dies and the base die are connected using through silicon vias (TSVs). The base die mainly consists of I/O circuits and components to support testing and debugging. As shown in Figure 2(a), a DRAM die has two channels. Each channel consists of bank groups and TSV I/O circuitry. A bank group contains several banks (e.g., 2 banks). Within a bank, there are DRAM cell arrays with sense amplifiers, row and column decoders, I/O sense amplifiers (IOSAs), and write drivers (WTD). To support near-data computing, a portion of banks are equipped with CIM units (Figure 2(b)), each of which contains SRAM buffers and processing elements (PEs).

C. Pull-based and Push-based Dataflows

There are two typical strategies for the aggregation phase, pull-based dataflow and push-based dataflow. As shown in Figure 3(a), with pull-based dataflows, each vertex (e.g., V_1) will pull the features of its neighboring vertices to perform computation. From the matrix perspective, the access to rows in matrix X^kW^k is random due to the uncertain distribution of non-zero elements in matrix $\hat{A}$. In contrast, each row of matrix X^{k+1} is accessed sequentially for the row-by-row accumulation. Therefore, the data reuse of matrix X^kW^k is poor, while that of matrix X^{k+1} is good. Push-based dataflows broadcast the features of each vertex to its neighbors, as shown in Figure 3(b). Thus, each row of matrix X^kW^k only needs to be loaded once. However, to accumulate the partially aggregated features, matrix X^{k+1} will be accessed randomly. Both two dataflows can also be adopted by the combination phase for its SpMM kernel.

D. Motivation

Data movement takes up most of the system energy consumption. In 3D CIM architecture, the data movement issue mainly

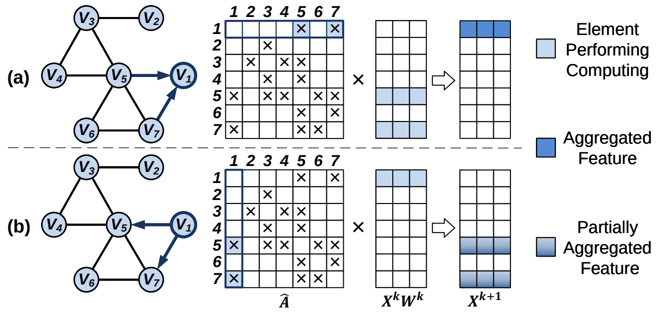

Fig. 3: (a) Pull-based dataflow. (b) Push-based dataflow

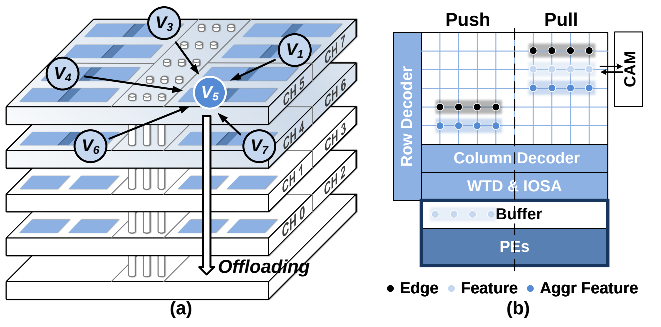

Fig. 4: A motivational example to illustrate data movement issues.

comes from two aspects: unoptimized mapping and dataflow. For the purpose of illustration, we use a motivational example to explain this issue. For mapping, aggregation operations are mapped to the CIM units to enjoy considerable DRAM bandwidth. However, as shown in Figure 4(a), the high-degree vertices (e.g., V_5) will gather all their neighbors from different banks and even different channels during the aggregation phase. This incurs significant data movements. For dataflow, a pull-based aggregation strategy is adopted to take advantage of large DRAM capacity. Due to the uncertain graph topology, features of each vertex will be accessed dynamically. The addresses of cross-bank features need to be recorded by content addressable memory (CAM), as shown in Figure 4(b). Therefore, the limited capacity of CAM will lead to frequent fetching operations.

We argue that a hybrid mapping and a push-based strategy can effectively solve the data movement issue. First, by offloading the high-degree vertex to the base die, the cross-channel data movement could be significantly reduced. Second, by adopting a push-based aggregation strategy, features of each vertex will be moved to a bank once according to Section II-C. This requires further both architectural and system software supports. These observations motivate us to propose a hardware and software co-design approach to exploit the emerging 3D CIM architecture.

III. Lift: An Energy-Efficient 3D CIM Accelerator for GCNs

A. System Overview

This paper presents *Lift*, a hardware and software co-design GCN accelerator based on the 3D CIM architecture. The objective is to minimize data movement and reduce overall energy consumption. At the hardware level, we propose a lightweight processing unit (LPU) with a push-based dataflow for each CIM bank group. To process high-degree vertices efficiently, we present an auxiliary processing unit (APU) at the base die. At the software level, we adopt a hybrid mapping strategy to exploit potential data locality and fully utilize the valuable computing resources.

B. Architecture Design

Based on the conventional 3D CIM architecture (Figure 2), we modify its architecture and add dedicated logic to address the data movement issue for GCNs. Specifically, We introduce two dedicated processing units, including an LPU for each CIM bank group and an APU at the base die.

1) Lightweight Processing Unit: Typically, the capacity of each bank group is large enough to buffer over 10^5 feature vectors. Thus, we introduce LPUs for bank groups to enjoy considerable DRAM capacity and reduce data movement. Figure 5 illustrates the hardware design of an LPU. Each LPU consists of an input vector buffer, a look-ahead FIFO, a control unit, and multiply-accumulate (MAC) arrays for matrix operations. The look-ahead FIFO is a customized scratchpad memory and is used to buffer sparse matrix and hide latency. The input vector buffer is deployed to buffer input vectors, including rows of weight matrix W^k or matrix W^kX^k.

LPUs process the SpMM kernel using push-based dataflow. Compressed sparse column (CSC) format is adopted for the sparse matrix to adapt the push-based dataflow. To process the SpMM, each LPU will read a portion of the sparse matrix from banks within the current bank group and buffer these data at the look-ahead FIFO. After that, the column index of the sparse matrix will be obtained and used to prefetch the corresponding input vector from non-CIM bank group. The fetched input vector will be transferred through TSV bus and bank group bus. They are finally stored at the input vector buffer. Since each input vector can be fully reused, it only needs to be temporarily buffered rather than stored in memory banks. Then, MAC arrays will multiply each input vector by the non-zero elements of the sparse matrix and generate partially accumulated vectors. These vectors will be written to I/O sense amplifiers (IOSAs) that are connected with MAC arrays, and flushed back to memory bank cells. After the execution of the SpMM, each output vector that is fully accumulated will be written back to non-CIM bank groups, waiting for further processing.

2) Auxiliary Processing Unit: Lift introduces an APU at the base die to process the high-degree vertices. Although only a small portion of vertices are with a high degree, they usually incur severe cross-bank-group and even cross-channel data movement. It is worse for cross-channel data movement. Data will be transferred from one channel to the base die and then further moved to the target channel, which needs to move the data twice. Thus, Lift offloads high-degree vertices to the base die to alleviate such data movement.

The hardware design of APU is shown in Figure 6. APU mainly consists of MAC arrays and dedicated buffers. High-degree vertices are cache-friendly for their number is much

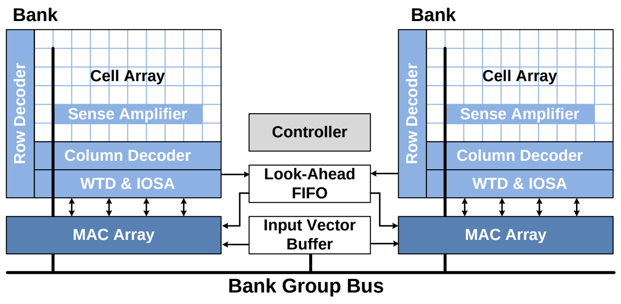

Fig. 5: A near-bank processing unit for a bank group.

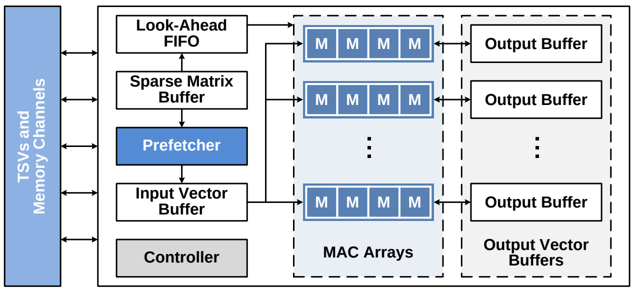

Fig. 6: The auxiliary processing unit at the base die.

lower than that of other vertices. Moreover, vertices with high degrees are also highly reusable since their output vector (or partially accumulated vectors) will be used as many times as their degree. Due to these factors, multiple output buffers are deployed to provide considerable on-die bandwidth and fully reuse the output vectors. Only a few of the dedicated buffers are used to temporarily cache the sparse matrix and input vectors.

APU adopts the push-based dataflow to reuse input vectors, as high-degree vertices usually have many common neighbors. Thus, the workflow of APU is similar to that of LPU. To process the SpMM, APU will fetch non-zero elements of the sparse matrix and prefetch input vectors from DRAM dies. Then, the input vector buffer will broadcast vectors to MAC arrays to perform vector-scalar multiplications. The partially accumulated vectors will temporarily store in their corresponding buffer. After the execution of the SpMM, each output vector will be flushed back to DRAM.

C. Hybrid Mapping

Although Lift introduces LPU and APU for the efficient processing of GCNs, there still lacks a mapping strategy to fully utilize hybrid computing resources provided by the 3D CIM architecture. Lift presents a hybrid mapping to jointly optimize both data locality and hardware utilization to reduce data movement and energy consumption. Vertices with strong connections can be mapped to LPUs for in-memory data reuse, whereas vertices with high-degree can be mapped to APU to alleviate cross-channel data movement. The hybrid mapping is a two-stage approach, where an initial mapping will be generated and then reallocated to reduce overall processing latency.

1) Generating an Initial Mapping: The initial mapping will decide which types of processing units vertices map to according to the computing capability. Assume that the computing capability of LPUs and APU are cap_L and cap_A , respectively; and the number of input graph's edges and vertices are num_E and num_V, respectively. Then, the threshold of vertex degree thr_d can be obtained, where the total edge number of vertices with degree greater than thr_d just exceeds:

$$exp_A = \frac{cap_A}{cap_L + cap_A} \cdot num_E \tag{2}$$

Vertices with degrees greater than thr_d will be mapped to APU, while vertices with degrees less than or equal to thr_d will be mapped to LPUs. To fully utilize the parallelism of CIM bank

Algorithm 1 Generating an initial mapping for LPUs

```
Require: The expected number of edges assigned to each LPU $exp_L$
    and the initial search depth of BDFS $init\_depth$.
Ensure: LPUs' mapping result $LPU[\,]$.
 1: $id \leftarrow 0$
 2: for $u \in V$ do
 3:     if $visit[u] \neq true$ then
 4:         BDFS$(u, init\_depth, id)$
 5:         if $LPU[id].edge \geq exp_L$ then
 6:             $id \leftarrow id + 1$
 7: function BDFS$(u, depth, id)$
 8:     if $depth > 0$ and $LPU[id].edge < exp_L$ then
 9:         $visit[u] \leftarrow true$
10:         if $u.degree \leq thr_d$ then
11:             Assign$(LPU[id], u)$
12:         for $v \in$ Neighbor$(u)$ do
13:             if $visit[v] \neq true$ then
14:                 BDFS$(v, depth - 1, id)$
15: end function
```

groups, Lift defines the expected number of edges assigned to each LPU:

$$exp_L = \frac{cap_L \cdot num_E}{(cap_L + cap_A) \cdot num_L} \tag{3}$$

where num_L is the number of LPUs.

To map vertices among LPUs without destroying data locality, bounded depth-first search (BDFS) is applied for the input graph. Algorithm 1 illustrates the generating of an initial mapping for LPUs. Lift will traverse each vertex in a depth-first manner. Vertices with degrees less than or equal to thr_d will be mapped to the current LPU $LPU[id]$ (Lines 10-11). Once the number of edges assigned to the current LPU exceeds exp_L, the remained vertices will be assigned to the next LPU (Lines 5-6). After BDFS-based traversal, all vertices are mapped to different LPUs.

2) Reallocation using Linear Programming: The initial mapping only considers the computing capability of processing units in spite of the latency of memory access. Such a mapping will cause an unbalanced workload and increase the overall processing latency and energy consumption. Thus, Lift applies the reallocation to further minimize the processing latency.

To measure the latency, we introduce a simplified performance model as follows:

$$\begin{cases} APU.t = T_A(APU.mem, APU.cmp) \\ LPU[i].t = T_L(LPU[i].mem, LPU[i].cmp) \end{cases} \tag{4}$$

where $APU.mem$ and $APU.cmp$ are the volumes of memory access and computation for APU, respectively; and function $T_A()$ is used to estimate APU's processing latency $APU.t$ using the sum of APU's memory access latency and computation latency. Similarly, $LPU[i].t$ is the estimated processing latency of i-th LPU. To avoid unbalanced traffic caused by memory access, Lift uses a round-robin based method to map the feature matrix and other data evenly among non-CIM bank groups. Thus, Lift adopts the average latency to estimate each memory access.

Lift reallocates edges and vertices assigned to each processing unit using linear programming to minimize the latency. We use $x_i (i \leq num_L)$ and x_A to decide how many edges should be reallocated between LPUs and APU, where there is $\sum_i x_i + x_A = 0$. Cost functions $C_A()$ and $C_i()$ are used to estimate the average latency change due to the reallocation per edge for i-th LPU and APU, respectively. To formulate this problem, we use latency $\mathbb{L}$ to denote the maximum latency between all processing units. Then, this problem can be solved efficiently using the following linear program:

$$\begin{array}{ll} \text{minimize} & \mathbb{L} \\ \text{subject to} & \sum_i x_i + x_A = 0 \\ & APU.t + C_A(x_A) < \mathbb{L} \\ & \forall i : LPU[i].t + C_i(x_i) < \mathbb{L} \end{array} \tag{5}$$

For i-th LPU with x_i less than zero, it needs to offload several vertices with the highest degree to APU until x_i edges are offloaded. Similarly, i-th LPU with x_i greater than zero will be reassigned vertices from APU.

IV. Evaluation

A. Experimental setup

Workloads. As shown in Table I, five frequently used graph datasets are adopted in the evaluation. A set of representative GCN applications are used, including GCN [1], GraphSage (GS) [12] and GINConv (GIN) [13]. Each GCN application has two layers with a hidden size of 128.

Simulation and Synthesis. To evaluate the performance of our approach, we modify and extend DRAMSim3 [14], a cycle-accurate memory simulator, to support CIM operations for GCNs. We measure the power, area, and delay of each component for simulation. Computing logic is implemented in Verilog and synthesized using Synopsys Design Compiler with the 90 nm library. CACTI [15] is adopted to estimate scratchpad memory, interconnect components, and 3D-stacked DRAM. All these components are converted to 32 nm technology.

Baselines. We compare Lift with HyGCN [4] and GCIM [10]. HyGCN [4] is a typical GCN accelerator with two hybrid processing engines, while GCIM [10] is a 3D CIM architecture with a pull-based dataflow for GCNs. Therefore, HyGCN and GCIM are selected for comparison. To evaluate the hybrid mapping scheme, we compare Lift with Lift-D. Lift-D only adopts LPUs. The hardware configuration of Lift is shown in Table II. For a fair comparison, HyGCN, GCIM, Lift-D, and Lift use 8 GBytes 3D-stacked memories with 256 GBytes/s off-chip bandwidth. HyGCN and GCIM both adopt 2,304 MACs as described in [10], while Lift-D and Lift both adopt 768 MACs.

B. Results and Discussion

1) Power and Area: Table III shows the power and area of two computing units, LPU and APU. These extended computing units introduce extra 2.9 W power consumption. An area overhead of 0.1098 mm^2 for each LPU takes up only 1.1% of a bank group's area. APU accounts for 1.7% of an HBM's area (i.e., 158 mm^2). Lift will not introduce extra area overhead for the base die since there is enough area budget [16].

TABLE I: Graph datasets.

Dataset	# of vertices	# of edges	# of features	# of classes
Citeseer (CS)	3,327	9,104	3,703	6
Cora (CR)	2,708	10,556	1,433	7
DBLP	17,716	105,734	1,639	4
Pubmed (PB)	19,717	88,648	500	3
Reddit (RD)	232,965	114,615,892	602	41

TABLE II: Hardware configuration of Lift.

	Configuration
HBM Organization	16 channels per stack, 8 bank groups per channel, 4 banks per bank group, 32 CIM bank groups
HBM Timing (ns)	tRCD = 10, tRAS = 17, tRC = 25, tCAS = 14 tRP = 10, tRRD = 5
LPU	1 KBytes look-ahead FIFO, 2 KBytes input vector buffer, 16 MACs (500 MHz)
APU	8 KBytes sparse matrix buffer, 16 KBytes input vector buffer, 512 KBytes output buffer, 256 MACs (500 MHz)

TABLE III: The power and area of different logic modules.

Module	Component	Power (mW)	Area (mm^2)
LPU	MAC Arrays	30.85	0.0659
	Input Vector Buffer	7.77	0.0251
	Look-Ahead FIFO	6.99	0.0188
APU	MAC Arrays	493.73	0.5273
	Dedicated Buffers	961.37	2.1837
	Others	23.66	0.0213

2) Energy Consumption: Figure 7 presents the normalized energy consumption of different benchmarks. From the experimental results, Lift can significantly reduce energy consumption. This is mainly due to the adoption of the CIM architecture. The proposed Lift can achieve 6.33×, 6.05×, and 1.25× energy savings compared to HyGCN, GCIM, and Lift-D, respectively. Lift also applies APU to alleviate data movement caused by high-degree vertices. This helps further reduce energy consumption.

3) Speedup: Figure 8 presents the speedup of different benchmarks. The proposed Lift can achieve average speedups of 8.69×, 4.43×, and 1.06× compared to HyGCN, GCIM, and Lift-D, respectively. This is due to adopting near-bank computing units with the push-based dataflow. This can help reduce the processing latency caused by data movement. For GS applications, Lift does not outperform Lift-D. This is because vertices are sampled, and there is no high-degree vertex.

4) Energy Breakdown: Figure 9 shows the energy consumption breakdown for GCIM, Lift-D and Lift, where labels "G", "D"

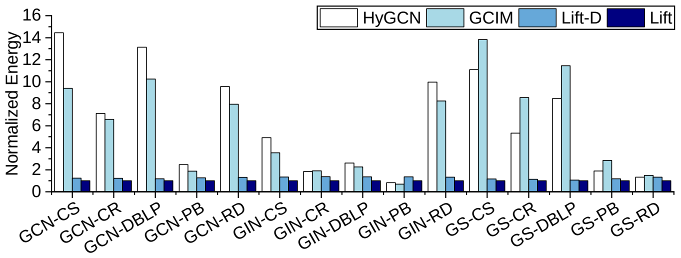

Fig. 7: Energy consumption for HyGCN [4], GCIM [10], Lift-D, and Lift.

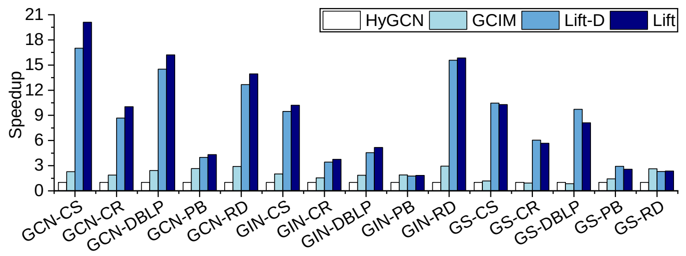

Fig. 8: Speedup for HyGCN [4], GCIM [10], Lift-D, and Lift.

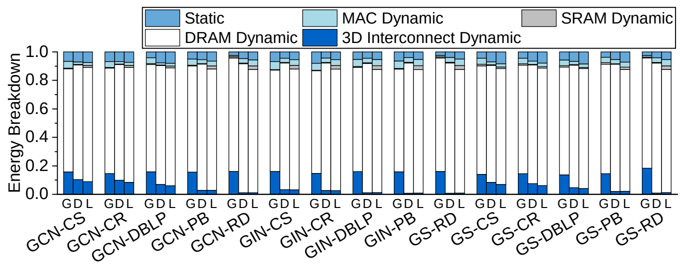

Fig. 9: Energy breakdown for GCIM [10], Lift-D, and Lift.

and "L" represent these three schemes, respectively. Dynamic DRAM has the highest proportion of energy consumption. GCIM, Lift-D, and Lift consume 15.44%, 3.85%, and 3.80% of the total energy for dynamic 3D interconnect energy, respectively. Lift has the least consumption of dynamic interconnect energy. This is mainly due to the joint optimization of the dataflow and mapping for the CIM architecture.

5) Data Movement: This section presents the experimental results for the volume of data movement. As shown in Figure 10, Lift has the least data movement among these schemes. Compared to GCIM, Lift-D and Lift reduce average data movement by 84% and 86%, respectively. The performance gain mainly comes from the adoption of push-based dataflow and hybrid mapping, both of which can help reduce data movement.

V. Conclusion

This paper presents Lift, a hardware and software co-design approach to exploit the energy-efficient processing of GCNs on the 3D-stacked CIM architecture. At the hardware level, Lift optimizes the system architecture by integrating dedicated computing units for vertices with different characteristics. At the software level, Lift presents a hybrid mapping to leverage hybrid computing resources and exploit potential data locality. This joint optimization effectively captures the execution pattern of GCNs and fully utilizes the computing capability of the CIM architecture. The experimental results show that Lift can significantly reduce data movement and energy consumption compared with representative schemes. In the future, we plan to work on parameterized compiler design space exploration and obtain accurate execution patterns of GCN applications.

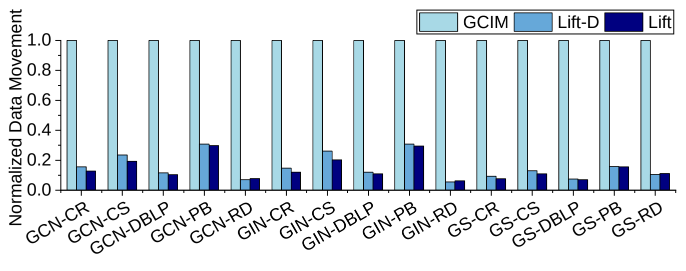

Fig. 10: Data movement for GCIM [10], Lift-D, and Lift.

VI. Acknowledgment

This work was supported in part by NSFC (61972259, 62072311, 62122056, 62102263, and U2001212), in part by Guangdong Basic and Applied Basic Research Foundation (2019B151502055, 2020B1515120028, 2022A1515010180), in part by Shenzhen Science and Technology Program (RCJC20221008092725019, JCYJ20210324094402008, JCYJ20210324094208024, and 20220810144025001), in part by Tencent "Rhinoceros Birds" - Scientific Research Foundation for Young Teachers of Shenzhen University. Yi Wang is the corresponding author.

References

[1] T. N. Kipf and M. Welling, "Semi-Supervised Classification with Graph Convolutional Networks," in *ICLR*, 2017, pp. 1–14.
[2] H. Wang, K. Wang, J. Yang, L. Shen, N. Sun, H.-S. Lee, and S. Han, "GCN-RL Circuit Designer: Transferable Transistor Sizing with Graph Neural Networks and Reinforcement Learning," in *DAC*, 2020, pp. 1–6.
[3] T. Geng, A. Li, R. Shi, C. Wu, T. Wang, Y. Li, P. Haghi, A. Tumeo, S. Che, S. Reinhardt, and M. C. Herbordt, "AWB-GCN: A Graph Convolutional Network Accelerator with Runtime Workload Rebalancing," in *MICRO*, 2020, pp. 922–936.
[4] M. Yan, L. Deng, X. Hu, L. Liang, Y. Feng, X. Ye, Z. Zhang, D. Fan, and Y. Xie, "HyGCN: A GCN Accelerator with Hybrid Architecture," in *HPCA*, 2020, pp. 15–29.
[5] S. Mondal, S. D. Manasi, K. Kunal, and S. S. Sapatnekar, "GNNIE: GNN Inference Engine with Load-Balancing and Graph-Specific Caching," in *DAC*, 2022, pp. 565–570.
[6] Z. Zhou, B. Shi, Z. Zhang, Y. Guan, G. Sun, and G. Luo, "BlockGNN: Towards Efficient GNN Acceleration Using Block-Circulant Weight Matrices," in *DAC*, 2021, pp. 1009–1014.
[7] T. Geng, C. Wu, Y. Zhang, C. Tan, C. Xie, H. You, M. Herbordt, Y. Lin, and A. Li, "I-GCN: A Graph Convolutional Network Accelerator with Runtime Locality Enhancement through Islandization," in *MICRO*, 2021, pp. 1051–1063.
[8] C. Chen, K. Li, Y. Li, and X. Zou, "ReGNN: A Redundancy-Eliminated Graph Neural Networks Accelerator," in *HPCA*, 2022, pp. 429–443.
[9] J. Li, A. Louri, A. Karanth, and R. Bunescu, "GCNAX: A Flexible and Energy-Efficient Accelerator for Graph Convolutional Neural Networks," in *HPCA*, 2021, pp. 775–788.
[10] J. Chen, Y. Lin, K. Sun, J. Chen, C. Ma, R. Mao, and Y. Wang, "GCIM: Toward Efficient Processing of Graph Convolutional Networks in 3D-Stacked Memory," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 3579–3590, 2022.
[11] S. Lee, S.-h. Kang, J. Lee, H. Kim, E. Lee, S. Seo, H. Yoon, S. Lee, K. Lim, H. Shin, J. Kim, O. Seongil, A. Iyer, D. Wang, K. Sohn, and N. S. Kim, "Hardware Architecture and Software Stack for PIM Based on Commercial DRAM Technology: Industrial Product," in *ISCA*, 2021, pp. 43–56.
[12] W. L. Hamilton, Z. Ying, and J. Leskovec, "Inductive Representation Learning on Large Graphs," in *NIPS*, 2017, pp. 1024–1034.
[13] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How Powerful are Graph Neural Networks?" in *ICLR*, 2019, pp. 1–17.
[14] S. Li, Z. Yang, D. Reddy, A. Srivastava, and B. Jacob, "DRAMSim3: A Cycle-Accurate, Thermal-Capable DRAM Simulator," *IEEE Computer Architecture Letters*, pp. 106–109, 2020.
[15] K. Chen, S. Li, N. Muralimanohar, J. H. Ahn, J. B. Brockman, and N. P. Jouppi, "CACTI-3DD: Architecture-Level Modeling for 3D Die-Stacked DRAM Main Memory," in *DATE*, 2012, pp. 33–38.
[16] X. Xie, Z. Liang, P. Gu, A. Basak, L. Deng, L. Liang, X. Hu, and Y. Xie, "SpaceA: Sparse Matrix Vector Multiplication on Processing-in-Memory Accelerator," in *HPCA*, 2021, pp. 570–583.