S-LGCN: Software-hardware co-design for accelerating LightGCN

Shun Li[1, 2, *], Ruiqi Chen[1, 3, *], Enhao Tang[1], Yajing Liu[2], Jing Yang[2], Kun Wang[1, †]
[1]State Key Lab of ASIC & System, Fudan University, Shanghai, China
[2]College of Physics and Information Engineering, Fuzhou University, Fuzhou, China
[3]VeriMake Innovation Lab, Nanjing, China
† kun.wang@ieee.org

***Abstract*—Graph Convolutional Networks (GCNs) have garnered significant attention in recent years, finding applications across various domains, including recommendation systems, knowledge graphs, and biological prediction. One prominent GCN-based recommendation model, LightGCN, optimizes embeddings for final prediction through graph convolution operations, and has achieved outstanding performance in commodity recommendation and molecular property prediction. However, LightGCN suffers from suboptimal layer combination parameters and limited nonlinear modeling capabilities on the software side. On the hardware side, due to the irregularity of the aggregation phase of LightGCN, CPU and GPU executions are not efficient, and designing a accelerator will be constrained by transmission bandwidth and the efficiency of the sparse matrix multiplication kernel. In this paper, we optimize the layer combination parameters of LightGCN by Q-learning and add hardware-friendly activation function to enhance its nonlinear modeling capability. The optimized LightGCN not only performs well on the original dataset and some molecular prediction tasks, but also does not incur significant hardware overhead. Subsequently, we propose an efficient architecture to accelerate the inference of LightGCN to improve its adaptability to real-time tasks. Comparing S-LGCN to Intel(R) Xeon(R) Gold 5218R CPU and NVIDIA RTX3090 GPU, we observe that S-LGCN is 1576.4× and 21.8× faster with energy consumption reductions of 3211.6× and 71.6×, respectively. Compared to FPGA-based accelerator, S-LGCN demonstrates 1.5-4.5× lower latency and 2.03× higher throughput.**

I. Introduction

The necessity to process graph data in real-world scenarios has led to the advancement of Graph Neural Networks (GNNs), with Graph Convolutional Neural Networks (GCNs) being particularly designed to learn intricate node features within graph structures [1]. This technology has found widespread adoption in diverse fields such as social networks, knowledge graphs, and biological prediction. LightGCN [2] is a GCN-based recommendation model that achieves higher performance with a simple and elegant structure. Nowadays, LightGCN plays an important role in personalized recommendation [3] and molecular property prediction [4].

Despite having shown considerable potential, there are two aspects of LightGCN that need further optimization. First, the model's layer combination parameters are not optimized, and some of the nonlinear modeling capabilities are lost. For the optimization of algorithm, past research has improved LightGCN's performance by incorporating supervised learning [4] or optimizing negative sampling [5]. These approaches significantly increase the computational burden during training and complicate the model, making it less hardware-friendly. In contrast, we employ a Q-learning-based method to optimize layer connection parameters and enhance the model's expressive power by incorporating hardware-friendly activation functions.

Another worrisome aspect is the time-consuming inference of LightGCN in the face of linear or polynomial growth interactions. To be more specific, in the task of predicting the interaction between lnc-RNAs and drugs, the computational complexity of LightGCN will grow polynomially if the base of lnc-RNAs and the diversity of drugs increase simultaneously. As a result, the inference time of LightGCN offers significant opportunities for enhancement. Due to the irregularity of the aggregation phase, it is difficult for the CPU to efficiently utilize data between compute units, while the GPU is inherently optimized for compute-intensive workloads with regular execution modes and is inefficient in handling the aggregation phase with irregular memory accesses. Customizing the computational architecture specifically for LightGCN to accelerate its time-consuming phase represents an ideal approach. Field-Programmable Gate Arrays (FPGAs), with their programmability and enhanced parallelism, have risen to prominence as favored platforms for algorithm acceleration [6]. Notable examples include Boost-GCN [7] and I-GCN [8]. These accelerators do not explore inter-layer parallelism since there are no layer combination in GCN, resulting in their architecture that is not efficient for inference of LightGCN. However, accelerating the time-consuming phase of LightGCN on FPGA is not without its challenges: 1) Data Storage and Memory Access: Being a memory-intensive application, LightGCN's computational latency will be constrained by transmission bandwidth; 2) Sparse-Dense Matrix Multiplication: Achieving efficient processing of Sparse-Dense Matrix Multiplication within LightGCN requires adept handling of potential parallelism especially inter-layer parallelism, and careful elimination of data dependencies. These constitute key challenges that demand meticulous attention in the pursuit of FPGA-based acceleration for LightGCN.

Our research endeavors to enhance LightGCN through two algorithmic optimizations. First, we deploy Parameter Exploration experiments based on Q-learning [9] to derive layer connection parameters that offer improved interpretabil-

[*] These authors contributed equally to this work. [†] Corresponding author.
This work was financially supported in part by National Key Research and Development Program of China under Grant 2021YFA1003602, and in part by Shanghai Pujiang Program under Grant 22PJD003.

979-8-3503-4859-0/DATE24/© 2024 EDAA

ity. Second, we introduce a nonlinear function at the output stage of each layer, elevating the model's nonlinear modeling capability. Optimized LightGCN performs better on the dataset from the original work and demonstrated better performance in two interaction prediction tasks (lncRNA-drug and lncRNA-disease). In addition to algorithmic improvements, we propose S-LGCN to accelerate the time-consuming phase of LightGCN. S-LGCN utilizes High Bandwidth Memory (HBM) to store data and alleviates the transmission bandwidth bottleneck through efficient data compression format and data reuse. In addition, S-LGCN performs three levels of parallel computation in processing sparse matrix multiplications, which greatly improves the inference speed. In summary, our work makes the following major contributions:

Hardware-friendly algorithm optimization. We optimize the layer combination parameters of LightGCN and add hardware-friendly activation functions with little hardware overhead, achieving better results on datasets from different domains.

Efficient architecture. We propose S-LGCN, an FPGA-based accelerator which includes a new data compression format, data reuse and three levels of parallelism.

Adaptable to real-time tasks. We accelerate the time-consuming process of LightGCN with a performance improvement of 1576.4× and 21.8× over CPUs and GPUs, respectively, making the model more adaptable to real-time tasks and meeting real-world demands. To the best of our knowledge, S-LGCN is the first hardware accelerator for LightGCN.

II. Motivation

A. For higher algorithm performance

GSLRDA [4] implements supervised learning of LightGCN by three data augmentation methods: node loss, edge loss, and random walk. While the algorithm has demonstrated enhanced performance, this improvement is accompanied by the integration of complex computations, thereby compromising the model's initial simplicity. Implementing minor modifications to the foundational model to augment the efficacy of LightGCN in specific tasks would constitute a more advantageous approach. Since the importance of the results of each layer is not uniform, it is clearly not reasonable to set the parameters uniformly to accomplish the layer combination. Layer combination parameters should be related to the benefits that the layers brings to the final result. And some activation functions with low computational burden can be considered to be added to LightGCN, which can enhance some of the nonlinear modeling capabilities without losing simplicity.

B. For faster inference

Real-time recommendation [10] has had a profound impact on retail, media, entertainment and other industries. GCN-based recommendation models inherently involve time-consuming aggregation phases, thereby limiting their suitability for real-time recommendation tasks. As the volume of interactions increases, this limitation becomes increasingly pronounced. As an illustration, given an increase in the quantity of user-related information $O(N)$ and item-related information $O(M)$, the additional computational complexity incurred in the aggregation phase is $O(N*M)$. This underscores a distinct practical imperative: achieving faster inference by accelerating the aggregation phase via customized architecture. Critically, such an architecture requires more scalability to be compatible with diverse model configurations, ensuring adaptability for various recommendation tasks.

III. Algorithm Introduction and Optimization

A. Algorithm introduction

LightGCN utilizes the aggregation process of GCN to obtain higher-level node representations and obtains inputs for performing predictions through layer combination. For the time-consuming and more irregular embedding update phase, it can be represented in matrix form. Let the 0-th layer embedding matrix be $E^{(0)}$, the adjacency matrix be A, we obtain the matrix equivalent form of Light Graph Convolution as:

$$E^{(k+1)} = \left(D^{-\frac{1}{2}} A D^{-\frac{1}{2}}\right) E^{(k)}$$

D is a diagonal matrix, in which each entry D_{ii} denotes the number of nonzero elements in the i-th vector of A. Normalized adjacency matrix is denoted as $\tilde{A}$. Lastly, the final embedding matrix used for prediction as:

$$\begin{aligned} E &= p_0 E^{(0)} + p_1 E^{(1)} + p_2 E^{(2)} + \cdots + p_k E^{(k)} \\ &= p_0 E^0 + p_1 \tilde{A} E^{(0)} + p_2 \tilde{A}^2 E^{(0)} + \cdots + p_k \tilde{A}^k E^{(0)} \end{aligned}$$

B. Hardware-friendly optimization

Algorithm 1 Parameter exploration based Q-learning

Constraint: $p_0 + p_1 + p_2 + \cdots + p_k = 1, p_0 \neq 0, step = 0.1$
Generate parameter list : $L^i = (p_0^i, p_1^i, p_2^i, \ldots, p_k^i)$
if $Result_{last} < Result_{current}$: **then**
  $reward = 1$
else
  $reward = -1$
end if
Q-table: Q(s, a) = 0; learning rate: lr=0.1; discount factor: γ=0.9;
delta = abs($Result_{current}$ - $Result_{last}$)
while delta $<$ 0.001 **do**
  $LightGCN\_train\_function(p_0^i, p_1^i, p_2^i, \ldots, p_k^i)$
  $reward$(s)
end while
$Q(s,a) \Leftarrow Q(s,a) + lr * \{[reward + \gamma * max\_a' [Q(s', a')] - Q(s,a)\}$
$s' \Leftarrow s$; output Q-table,state
return optimized parameters

Parameter exploration. Given the non-uniform importance of results from each layer, uniformly setting the layer combination parameter to $1/k + 1$ leads to missed opportunities for performance improvement. To overcome this limitation, we devised a Q-learning based parameter search experiment to achieve superior results compared to the original model. Through a thorough analysis of the original model, we formulated constraints for the parameter exploration to capture the self-connection effect. **Algorithm 1** presents the constraints and the parameter exploration. The goal for parameter exploration is to maximize either Normalized Discounted Cumulative Gain (NDCG) or Area Under the ROC Curve (AUC), depending on the desired performance metrics. We introduce

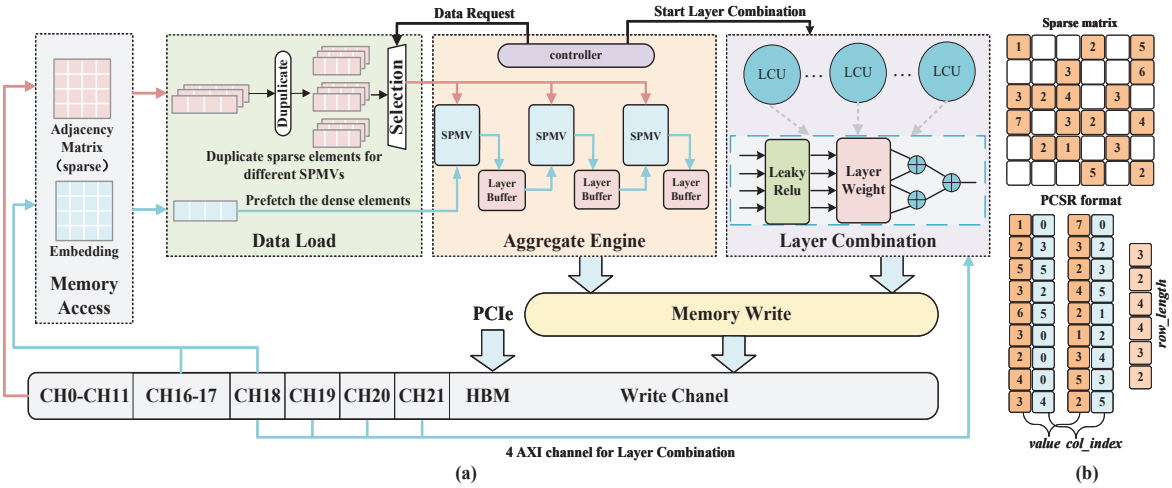

Fig. 1. (a) Overview of S-LGCN microarchitecture; (b) An example of PCSR compression format.

a convergence condition to terminate the exploration and output the optimized parameters when the absolute difference between two execution results is less than 0.001.

Output optimization. To enhance the capacity of the model for generalization and nonlinear representation, a LeakyReLU activation function is incorporated at the output stage of each layer. We use NSP to represent the negative slope, and LeakyReLU is defined as:

$$f(x) = max(0, x) + \alpha(NSP) \times min(0, x)$$

The incorporation of the LeakyReLU function in the output stage improves model tuning and representation, enabling the graph convolutional layer to move beyond mere linear mapping and acquire the capacity to learn intricate functional relationships. In comparison to alternative functions, LeakyReLU exhibits greater linearity, which accelerates the training process and simplifies the design of hardware circuits.

The two aforementioned optimizations not only improve the performance of LightGCN but are also judiciously attuned to hardware considerations. Specifically, optimizing layer combination parameters incurs no additional hardware overhead. Furthermore, the integration of LeakyReLU at the output stage constitutes a hardware-friendly modification. Its local linear operation renders it susceptible to hardware implementation via multiplier and multiplexer collaboration, incurring minimal incremental costs. Subsequent sections will elaborate on an accelerator architecture tailor-made for LightGCN.

IV. Microarchitecture

We propose S-LGCN, an efficient accelerator for optimized LightGCN. Fig. 1(a) illustrates four modules: memory access, data load, aggregate engine, and layer combination. The memory access module efficiently retrieves elements from the adjacency matrix and embedding matrix stored in the HBM. The data load module fetches data from HBM, aligning it with the computational strategy of the aggregate engine. The aggregate engine concurrently processes multiple matrix-vector multiplications, utilizing both parallel and pipelined designs. The layer combination module implements the fusion of the results of various layers.

A. Memory access

When handling memory-intensive tasks like sparse matrix multiplication (SPMM), HBM has greater potential than DDR [11]. To store the sparse matrix A, we propose the PCSR compressed format, depicted in Fig. 1(b). In this format, the value represents the element's value, and the col index denotes its corresponding column index. These values are packed together in a 64-bit packet, each occupying 32 bits. The 64-bit row message is a compact packet that contains the row index along with the row length (the count of non-zero elements in a row), making use of the high data bits for row index storage and the remaining bits for row length. This addition of row index, in comparison to the typical CSR compression format, ensures data accuracy during parallel processing with a large number of processing elements (PEs). Notably, elements within the same row share a common row index, thus incurring only a minor memory overhead. Row message and all 64-bit packets belonging to that row are read out together, and we allocate 12 channels to support sparse element reading, using three channels to access the dense embedding matrix. This optimized access pattern achieves efficient memory access and helps improve the overall performance of the S-LGCN.

B. Data load

The data load module reads sparse element metadata from the on-chip buffer, structuring it into a comprehensive row-based task for the PE. Current dense vectors are preemptively stored in BRAM, enabling the PE to conduct multiplications by matching elements in BRAM based on non-zero column indices. Additionally, duplicating sparse element during read-out in the data load Module enables all SPMVs to exploit the present sparse element, facilitating inter-layer parallelism and augmenting data reusability. This shared optimization strategy mitigates data dependencies, minimizes redundant data transmissions, and consequently enhances the overall efficiency of the S-LGCN.

C. Aggregate engine

The aggregate engine consists of three sparse matrix-vector (SPMV) modules and on-chip buffers dedicated to embedding updates at each layer. Ensuring the scalability of architecture,

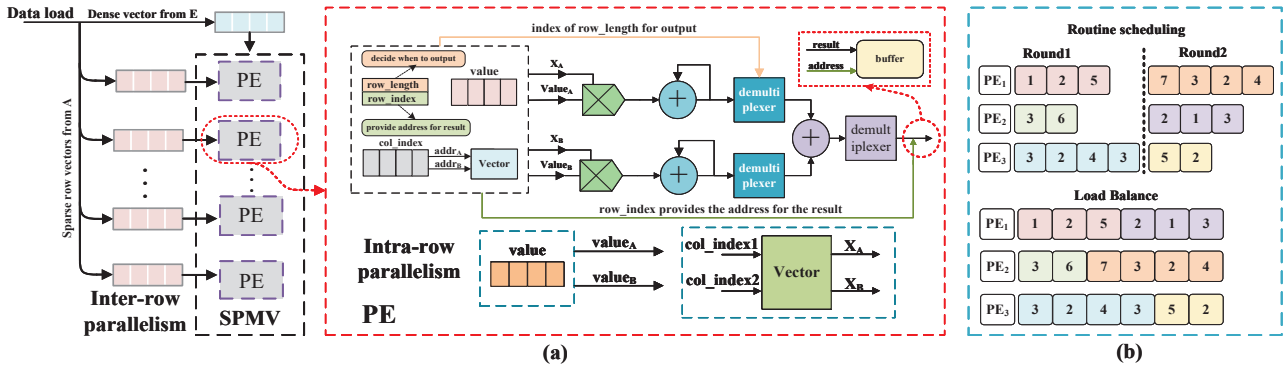

Fig. 2. (a) SPMV architecture with intra-row parallelism and inter-row parallelism; (b) Workload Balance across multiple PEs.

the quantity of activated SPMVs depends on the layers of LightGCN. The controller initiates a data request signal to the data load module and orchestrates the execution of layer combination. Furthermore, to prevent computational units from remaining idle and address data dependency bottlenecks, load balance and a feature pipeline are implemented for efficient inter-layer parallel computation.

SPMV. As depicted in Fig. 2(a), each SPMV module comprises multiple PEs that concurrently perform multiplication of different rows of the adjacency matrix with the same embedding vector. Within one cycle, each PE obtains the values of two non-zero elements and their corresponding column indices. These column indices serve as addresses to access the dual-port RAM (holding the embedding vectors) and retrieve the corresponding embedding elements. Subsequently, the two non-zero elements are multiplied with their respective embedding elements and fed into separate accumulators. These two multiplication operations run in two parallel computational pipelines until the multiplication of a single row of sparse matrix elements is completed. At this point, the results from the two accumulators are combined to generate the final outcome. The row length indicates whether the row has been computed, and the row index is utilized as an address to write the final result to the BRAM used for storing the result vector. Each PE is pipelined, enabling it to compute two data elements from the same row simultaneously, facilitating in-row parallelism. By employing multiple PEs, the module can perform the multiplication of multiple rows concurrently, achieving inter-row parallelism. Setting the number of PE channels to 2 offers two benefits: 1) It helps avoid pipeline idle time; 2) It optimizes the utilization of dual-port BRAM to conserve on-chip memory resources.

Workload balance. To achieve efficient parallel computing, 48 PEs are utilized within the SPMV module to enable simultaneous operations on different rows. To balance the workload, optimizations are implemented, as shown in Fig. 2(b). The designed task scheduling module computes the cumulative sum of row lengths in each PE, prioritizing task assignments to PEs that completed the previous round of computation. The evil rows result in extended computational latency for a single PE, yet exert no adverse impact on the functioning of additional PEs, given that each is capable of independently executing row-based tasks. This load balancing strategy efficiently utilizes computational resources and reduces waste of time and hardware resources.

Feature pipeline. Inter-layer propagation in LightGCN requires generating new embeddings, and from a matrix computation perspective, updating the first column of the embedding matrix (i.e., the first feature of the embedding) enables its participation in the subsequent layer. On-chip storage is utilized to cache the intermediate results of this column, which are later written back to the HBM. The input data's total bandwidth cannot support simultaneous computation of the three SPMVs for the same layer. To address this, we designed an optimization method called feature pipeline (see Fig. 3). Take the three-layer LightGCN for example, SPMV1, SPMV2, and SPMV3 are employed to compute the embeddings of the first, second, and third layers, respectively. Sparse elements in the data load module are shared among all SPMVs which allows for efficient propagation of the embedding among the three SPMVs in vector form. To illustrate the benefits of feature Pipeline, we define one period as the completion of multiplying normalized adjacency matrix $\tilde{A}$ with an embedding vector. When SPMV1 computes the first vector of the embedding matrix for the first layer, then used by SPMV2 to compute the first feature of the second layer, and subsequently by SPMV3 to compute the first feature of the third layer. While SPMV2 computes the first feature, SPMV1 performs the computation of the second feature, and other features are computed with the same rule. Notably, the advantage of the feature pipeline becomes more pronounced as the number of vectors in the embedding matrix increases.

D. Layer combination

The layer combination (LC) module performs nonlinear activation and weighting of the results of different layers to obtain the final embedding. LC consists of multiple Layer Combination Units (LCUs) and starts working when an embedding vector passes through a complete feature pipeline, meaning E_i^0, E_i^1, E_i^2, and E_i^3 have been computed and written back to the HBM (i represents the index of a vector). To efficiently handle the data from the four AXI channels connected to the same mini switch in the HBM, we have dedicated data channels for each E_i^0, E_i^1, E_i^2, and E_i^3, avoiding unnecessary delays. The data from the four AXI channels first pass through the LeakyReLU module to achieve nonlinear activation. Afterward, the layer combination parameters obtained in the parameter exploration experiments were multiplied with elements from the corresponding layers. Finally, matrix

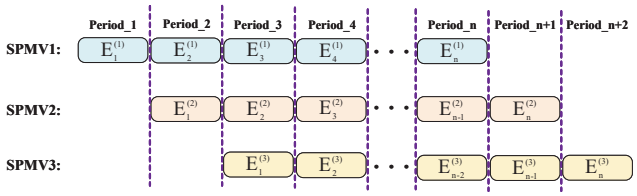

Fig. 3. Feature Pipeline Data Flow in Aggregate Engine.

addition is performed sequentially in the order of the columns to obtain the result of final embedding used for prediction.

V. Evaluation

A. Experiment setup

The proposed S-LGCN are implemented in Verilog HDL and synthesized using Vivado 2020.2 on a Xilinx FPGA Alveo U280 accelerator card running at 225 MHz. Data exchange between the CPU and FPGA is facilitated through the PCIe interface of the U280, with all data stored in the HBM. To evaluate algorithmic optimization, we make a parallel comparison with the original work and demonstrate the competitiveness of our optimized LightGCN on molecular property prediction. Hardware comparisons were performed against Intel(R) Xeon(R) Gold 5218R CPUs and NVIDIA RTX3090 GPUs. Furthermore, a comprehensive comparison was conducted between S-LGCN and BoostGCN.

B. Experiment for the optimizations

To validate the effectiveness of the optimizations, we conducted experiments with a set of weights obtained from the parameter search for optimizing the layer combination. LeakyReLU function was used in the output phase. The training results on the Lastfm dataset are illustrated in Fig. 4. To observe the training trend fully, we conducted training for 2000 epochs. The original LightGCN exhibits a significant decline in ndcg and recall. LightGCN converges faster due to the fact that it removes the activation function making the model simple, but inevitably overfitting occurs, mainly due to the inherent linear tendency of the model. The optimized LightGCN demonstrates an outstanding training curve, steadily increasing over 2000 epochs. The evaluation results on other datasets are shown in the TABLE I, LightGCN-P optimizes only the parameters of layer combination and LightGCN-L adds LeakyReLu. The optimized LightGCN demonstrates its advantages. Notably, the optimization effect is particularly pronounced for two datasets, Yelp2018 and Amazon-book, indicating that this optimization approach possesses good generalization capabilities. Additionally, we have conducted an assessment of the hardware overhead associated with the optimization. The hardware overhead induced by the optimization is predominantly evident in the LCU. TABLE II offers a comparative analysis between the LCU implementation in the original model and that in the optimized model. Additional LUT and DSP are used to implement the calculation of LeakyReLu, CYCLE represents the number of cycles for a single LCU to process a set of data. The parallelism of the LCU only needs to match the bandwidth of the 4 AXI channels. Consequently, the additional overhead brought by this hardware-friendly optimization is minimal.

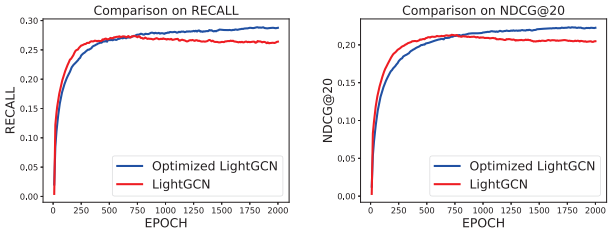

Fig. 4. Training curves of LightGCN before and after optimization.

TABLE I
Performance comparison of optimized LightGCN with other recommendation models.

Dataset	Lastfm		Gowalla		Yelp2018		Amazon-book	
Method	ndcg	recall	ndcg	recall	ndcg	recall	ndcg	recall
NGCF	N/A	N/A	0.1327	0.1570	0.0477	0.0579	0.0263	0.0344
Mult-VAE	N/A	N/A	0.1335	0.1641	0.0450	0.0584	0.0315	0.0407
GRMF	N/A	N/A	0.1205	0.1477	0.0462	0.0571	0.0270	0.0354
LightGCN	0.2095	0.2658	0.1555	0.1823	0.0525	0.0639	0.0318	0.0410
LightGCN-P	0.2168	0.2726	0.1554	0.1820	0.0534	0.0648	0.0328	0.0421
LightGCN-L	0.2101	0.2669	0.1551	0.1824	0.0530	0.0641	0.0322	0.0411
Optimized LightGCN	**0.2232**	**0.2872**	**0.1558**	**0.1827**	**0.0541**	**0.0655**	**0.0329**	**0.0423**

TABLE II
Hardware overhead from optimization.

Resource	LUT	FF	DSP	BRAM	CYCLE
LCU(original)	489	872	14	4	4
LCU(optimized)	519	892	22	4	5

C. Experiment for molecular property prediction

The evaluation results of biological tasks are shown in TABLE III. GSLRDA [4] uses LightGCN in the relationship prediction of lncRNA and drug and combines with supervised learning, and the performance outperforms some of the classical methods. GANLDA [12] is also a graph-based recommendation model for the relationship prediction of lncRNA and disease. For a fair comparison, the dataset and configuration of all experiments are kept consistent. In such tasks, AUC is considered a critical metric, we can think of AUC as the probability that the predicted value of positive samples is greater than the predicted value of negative samples in a set of tests. The optimized LightGCN shows competitive performance, which proves that the optimized LightGCN has better generalization and modeling capabilities.

TABLE III
Comparison with other graph-based models.

Dataset	lnc-RNA	drug/disease	interactions	experiment	method	AUC
R-drug	625	121	2693	5-fold	GSLRDA [4]	0.9101
					Ours	0.9204
R-disease	240	412	2697	10-fold	GANLDA [12]	0.8834
					Ours	0.9022

D. Comparison with FPGA based accelerator

S-LGCN represents the first work to accelerate LightGCN, making it difficult to execute a fair comparison with existing accelerators. A high-performing FPGA-based GCN accelerator called BoostGCN was evaluated on commonly used recommendation system datasets, Yelp and Amazon, which is similar to our work. Therefore, we compare the feature update time between S-LGCN and BoostGCN. It should be noted that the main major difference between the two is that BoostGCN contains the feature transformation while S-LGCN contains the layer combination. Even though this difference exists, the feature update time can represent the time that takes for a graph-based accelerator to obtain more

TABLE IV
COMPARISON OF S-LGCN WITH ADVANCED ACCELERATOR.

Platform	\	BoostGCN	S-LGCN
Frequency(MHz)	\	250	225
Resources	ALM/LUT	294K	90K
	DSP	3840	864
	BRAM	18MB	8MB
Latency(ms)	Yelp	193	42.5
	Amazon	793.5	533.5
Throughput(Gb/s)	\	6.496	13.184

advanced features. To ensure fairness, S-LGCN maintains the same feature dimensions and number of model layers, and is evaluated on the same dataset from [13]. TABLE IV presents a data comparison between S-LGCN and BoostGCN in terms of computational resources, latency, and throughput. S-LGCN demonstrates lower computational resource consumption and outperforms BoostGCN by 4.5× on the Yelp and 1.5 × on the Amazon dataset. BoostGCN's optimization of the execution order within the GCN layer reduces floating-point operations and memory accesses during the aggregation phase, contributing to its performance improvement. On the other hand, S-LGCN needs to complete the aggregation phase first, which results in a higher number of floating-point operations and memory access. Nevertheless, S-LGCN achieves 1.5× performance improvement with balanced PE workload and three levels of parallelism. Furthermore, S-LGCN achieves a 2.03× improvement in throughput rate compared to BoostGCN, reaching 13.184 Gb/s. This improvement is primarily attributed to HBM's high-bandwidth data transfer and inter-layer parallel computing based on feature pipeline.

E. Comparison with CPU and GPU

We conduct experiments with Optimized LightGCN on an Intel(R) Xeon(R) Gold 5218R CPU (CPU) and an NVIDIA RTX3090 GPU (GPU) using identical model configurations, Fig. 5 (a) illustrates the speed up ratio of various platforms, using CPU as the baseline. Impressively, S-LGCN achieves an average speed improvement of 1576.4× over CPU and 21.8× over GPU. This remarkable enhancement is a result of employing a customized dedicated datapath and three levels of parallel computing. We utilize analytical tools to measure the average power consumption of both CPU and GPU, subsequently utilizing equations to ascertain the energy efficiency across diverse platforms, energy efficiency= $1/(\text{Power} \times \text{Latency})$. Fig. 5 (b) illustrates the energy efficiency comparison between FPGA and GPU. The results indicate that the energy efficiency of S-LGCN significantly surpasses that of both CPU and GPU, an advantage attributable to its dedicated data pathways and the absence of redundant computations. Experimental results reveal that S-LGCN improves energy efficiency by an average of 3211.6 × and 71.6 × compared to CPU and GPU. The comprehensive evaluation of latency and energy efficiency corroborates that our proposed accelerator architecture markedly enhances the inference efficiency of LightGCN.

VI. CONCLUSION

In this paper, we first optimize the layer combination parameters of LightGCN and enhance its nonlinear modeling capability. Additionally, we propose S-LGCN, an FPGA-based

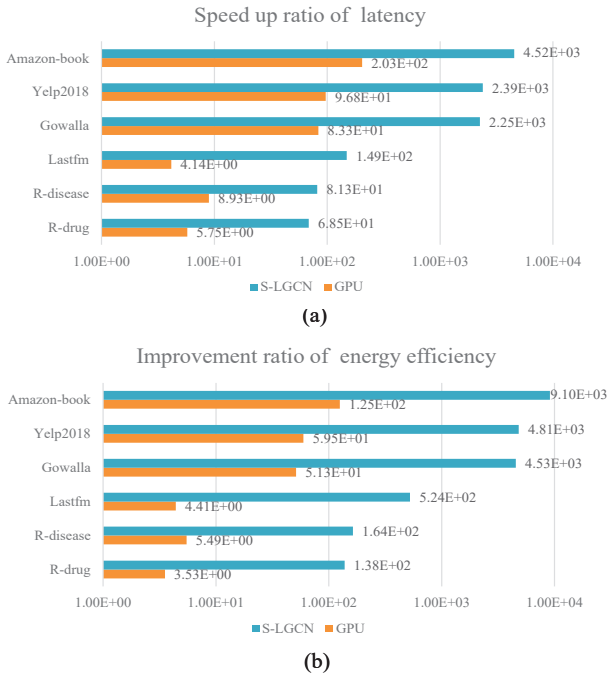

Fig. 5. Comparison with CPU and GPU(normalized over CPU). (a) Speedup ratio comparison; (b) Improvement ratio of energy efficiency comparison.

LightGCN accelerator. We employ a fully pipelined computational unit and incorporates three levels of parallelism to accelerate optimized LightGCN. S-LGCN performs competitively on the original datasets and the molecular property prediction. We demonstrate that the proposed accelerator architecture significantly enhances the inference speed of LightGCN, thereby better accommodating real-time recommendation tasks.

REFERENCES

[1] Wu, Z. *et al.*, "A comprehensive survey on graph neural networks," *TNNLS*, 2020.

[2] He, X. *et al.*, "Lightgcn: Simplifying and powering graph convolution network for recommendation," in *SIGIR*, 2020.

[3] Huang, W. *et al.*, "Dual-lightgcn: Dual light graph convolutional network for discriminative recommendation," *Computer Communications*, 2023.

[4] Zheng, J. *et al.*, "Graph neural network with self-supervised learning for noncoding rna–drug resistance association prediction," *JCIM*, 2022.

[5] Huang, T. *et al.*, "Mixgcf: An improved training method for graph neural network-based recommender systems," in *SIGKDD*, 2021.

[6] Yu, Y. *et al.*, "Opu: An fpga-based overlay processor for convolutional neural networks," *TVLSI*, 2019.

[7] Zhang, B. *et al.*, "Boostgcn: A framework for optimizing gcn inference on fpga," in *FCCM*, 2021.

[8] Geng, T. *et al.*, "I-gcn: A graph convolutional network accelerator with runtime locality enhancement through islandization," in *MICRO*, 2021.

[9] Watkins, C. J. *et al.*, "Q-learning," *Machine learning*, 1992.

[10] Ma, Y. *et al.*, "Temporal-contextual recommendation in real-time," in *SIGKDD*, 2020.

[11] Song, L. *et al.*, "Sextans: A streaming accelerator for general-purpose sparse-matrix dense-matrix multiplication," in *FPGA*, 2022.

[12] Lan, W. *et al.*, "Ganlda: graph attention network for lncrna-disease associations prediction," *Neurocomputing*, 2022.

[13] Zeng, H. *et al.*, "Graphsaint: Graph sampling based inductive learning method," in *ICLR*, 2020.