GPACE: An Energy-Efficient PQ-based GCN Accelerator with Redundancy Reduction

Yibo Du[1,3], Shengwen Liang[2,3], Ying Wang[1,3], Huawei Li[2,3,5], Xiaowei Li[2,3], Yinhe Han[1,3,4]
[1] CICS, Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China
[2] SKLP, Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China
[3] University of Chinese Academy of Sciences, Beijing, China
[4] Zhejiang Lab, Zhejiang, China [5]Peng Cheng Laboratory, Shenzhen, China
{duyibo21s, wangying2009, liangshengwen, lihuawei, lxw, yinhes}@ict.ac.cn

***Abstract*—Graph convolutional network (GCN) has been proven powerful in various tasks for it combines both neural networks and graph processing operators. However, this characteristic makes GCN exhibit hybrid execution patterns, which is unfavorable for CPUs and GPUs. Therefore, designing specialized GCN accelerators is becoming a prevalent paradigm. Unfortunately, as graph scale continues to grow, existing GCN accelerators suffer from significant bandwidth consumption and memory footprint as they neglect the inherent semantic redundancy of vertex features. Although applying Product Quantization to GCN is a promising solution to reduce the sizeable graph data via distilling semantic redundancy, it introduces novel operations with unique patterns that existing GCN accelerators cannot support.**

In this paper, we propose GPACE, an energy-efficient GCN accelerator that can fully harness the potential of PQ to reduce bandwidth consumption and data movement. GPACE is designed with a lookup-efficient architecture and well-optimized dataflow to support the unique data access and computation pattern of PQ-GCN. In addition to leveraging PQ to distill semantic redundancy, we exploit the operation redundancy and propose a redundancy-aware architecture to detect and reduce types of redundant operations to achieve higher energy efficiency. Evaluations show GPACE achieves high speedup and energy saving compared with CPU, GPU, and specialized GCN accelerators.

I. Introduction

To capture real-world relationships, organizing data in graph structures is increasingly prevalent. Graph Convolutional Network (GCN) [1] has emerged as a powerful approach to process graph data as it combines both traditional neural networks and graph processing operators. However, this combination brings a hybrid execution pattern to GCN, which makes general-purpose processors such as CPU and GPU suffer from inefficiency, especially the irregular data access introduced by sparse graph operations. To this end, designing specialized GCN accelerators [2]–[6] has been a popular designing paradigm.

Despite prior GCN accelerators achieve performance improvements over conventional processors, they still suffer from high memory footprint and bandwidth consumption due to large input graph data [7]. This comes from two aspects. First, real-world graphs contain a huge number of vertices. For example, in a social network, enormous users require large memory and computation resources to handle a such scale graph. A common dataset Reddit [8] requires 972MB memory, which challenges on-chip memory and bandwidth. Second, semantic information contained in vertex features is often represented in exhaustive high-dimension to ensure the information integrity as much as possible. Therefore, the sizable input graph data requires large memory capacity and makes existing hardware suffer from high bandwidth consumption, which limits the usage of GCN in resource and energy-constrained scenarios.

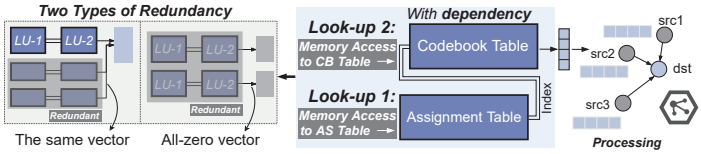

Fig. 1. Look-up chain exhibits unique data access and computation patterns.

To alleviate the high memory footprint and bandwidth consumption of large-scale graphs, reducing the input graph size is considered an effective approach, such as sparsity compression [9] and Product Quantization (PQ) [10]. Among those methods, PQ can achieve a remarkable compression ratio via distilling semantic redundancy, as features in neural networks contain redundant semantic information. While compression ratio of other methods is limited by data sparsity. For example, sparsity compression scheme underperforms in cases where feature density is high such as Reddit with more than 50% density and additional index increase overheads of the scheme. In contrast, the compression ratio of PQ is not limited by sparsity. Meanwhile, PQ is also possible to exploit sparsity acceleration like in sparsity compression. Therefore, PQ serves as an effective way to distill the semantic redundancy and reduce input data.

Unfortunately, applying PQ to reduce input data size introduces distinct operations to GCNs, specifically the ***look-up chain operation*** which makes existing hardware suffer from inefficiency due to its unique data access and computation patterns. As shown in Fig. 1, accessing feature vectors and performing operations on them requires two phases of look-up operations and calculations with dependencies. The first phase look-up is performed to access the index of the subvector. Then memory accesses to fetch corresponding codebooks are issued. The second phase look-up is performed to access the subvector using the retrieved index. This look-up chain exhibits a complex computation and data access pattern. **(1) Irregular look-up operation**. Look-up operations to the index in assignment table and centroid in codebook table exhibit high data access irregularity due to the random vertex access in graphs, which incurs frequent memory access. Besides, the randomness of index values in assignment table further exacerbates look-up irregularity in codebook. **(2) Unique computation pattern.** The introduced novel look-up operations and index calculations exhibit unique computation patterns with dependencies and require a specialized datapath and dataflow for efficient execution. In addition, look-up operations exhibit types of operation

979-8-3503-4859-0/DATE24/© 2024 EDAA

redundancy, as shown in Fig. 1. Experiments (Section III-B) show that redundant operations even account for up to 80% of the total. Therefore, it is necessary to reduce the redundant operations to achieve a higher energy efficiency.

However, the unique data access and computation patterns make existing GCN accelerators fail to support. First, look-up chain operations heavily utilize memoization. Existing GCN accelerators are not designed for look-up operations to manage the memoization results. Second, existing GCN accelerators lack the dedicated datapath to accommodate operations with dependencies and address types of operation redundancy. Therefore, a specialized accelerator is needed.

In this paper, we propose GPACE, an energy-efficient GCN accelerator that can fully harness the potential of PQ to reduce significant input-dominated bandwidth consumption and memory footprint. GPACE is designed with a look-up efficient architecture to support the look-up chain with unique computation and data access patterns. In addition, GPACE not only leverages PQ to distill the semantic redundancy but also can reduce types of operation redundancy. We propose a redundancy-aware look-up policy to detect and reduce the types of redundant operations to achieve higher energy efficiency. Last of all, we quantitatively optimize the dataflow to minimize the intermediate data access and exploit the available parallelism for an efficient acceleration. This paper makes the following contributions:

- We characterize the unique computation and data access patterns of PQ-GCN and propose GPACE, a novel GCN accelerator with specific architecture and efficient datapath that can fully harness the potentials of PQ to reduce the memory footprint and bandwidth consumption.
- We identify types of operation redundancy in PQ-GCN. Based on this, we propose a redundancy-aware policy and present a redundancy-aware architecture to detect and reduce redundant operations.
- We quantitatively optimize the dataflow to minimize the intermediate data access. Evaluated on real-world datasets, GPACE achieves 390.4×, 56.3×, 8.6× and 5.1× speedup, and 948.8×, 119.1×, 6.5×, and 4.1× energy savings on average over CPU-PyG, GPU-PyG, HyGCN, and ReGNN.

II. Background

Product Quantization encodes the vector into a compressed form, as shown in Fig. 2 left. The feature vector is divided into sub-vectors (blocks) with block size BS. Each sub-vector is mapped to the most similar centroid with an index. The centroids are stored in the Codebook Table learned by clustering algorithms. Therefore, the original feature vector can be reconstructed by the centroids. BS is set to 32 and the clustering algorithm clusters 1024 vectors to upper 64 learned centroids in this paper. Compared to other schemes, PQ can achieve an impressive compression ratio by distilling the semantic redundancy while not being limited to the sparsity. PQ is also possible to exploit sparsity acceleration like in sparsity compression schemes meanwhile. The PQ is performed only once during the training. Therefore, PQ is an appealing solution to distill the semantic redundancy and reduce data size.

PQ-GCN. The execution of PQ-GCN can be described as three stages in Fig. 2 right. The Look-up Chain stage performs two-phase look-up operations with operation dependencies to reconstruct original feature vectors. The Aggregation stage performs vector operations and neighbor aggregation on the reconstructed feature vectors. Then, the Feature Update stage produces a new state of the destination vertex using learned weights. The reconstruction of feature vectors needs two look-up operations. Specifically, the first look-up operation uses vertex ID to access indexes in assignment table. Then, the following look-up operations use retrieved indexes to access centroids in codebook table. The two-phase dependent look-up operations are denoted as the ***look-up chain***.

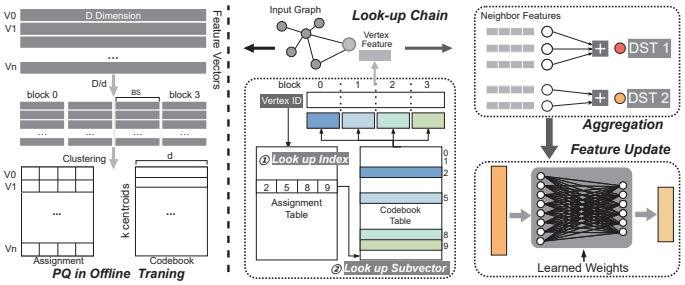

Fig. 2. Illustration of the PQ-based GCN model.

III. Motivation

A. Input-dominated Bandwidth and Memory Footprint

We conduct a quantitative analysis of GCN and find that GCN suffers from high bandwidth consumption and memory footprint due to the large input data. Fig. 3(a) illustrates the memory consumption breakdown of input vertex features and others including weights, intermediate data, and output. It can be observed that input vertex features contribute to substantial memory consumption. This can be attributed to two reasons. First, the number of vertices is tremendous in large-scale graphs such as a recommendation system where e-commerce data are on a billion-scale and require enormous resources. Second, vertices are often featured in a high dimension to get an exhaustive representation. Therefore, the data size of input vertex is much larger than that of other data, and vertex features dominate memory consumption. Besides, input vertex features in GCNs exhibit high bandwidth consumption. As shown in Fig. 3(b), input vertex features dominate the bandwidth. This is because, in addition to the large size of vertex features bringing massive data movement, vertex features exhibit low reusability. While weights have obvious reuse characteristics. Therefore, the input data can easily dominate memory bandwidth.

B. The Unique Pattern of PQ-GCN

Although GCN has gained superior performance in various tasks, the significant input-dominated bandwidth consumption and memory footprint limit its usage in energy and resources-constrained scenarios [11]. As discussed, PQ is an effective way to reduce input size. However, applying PQ to GCN introduces the distinct look-up chain with unique patterns that existing hardware fails to efficiently support.

The irregular data access. The look-up chain operations in PQ-GNN show irregular data access behavior. Firstly, due to the sparse and random edge distribution in real-world graphs, the first phase look-up operations to index using incoming neighbor IDs are irregular. Thereby, accessing subvector indexes in the

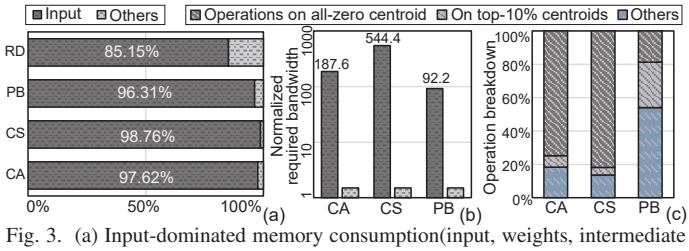

Fig. 3. (a) Input-dominated memory consumption(input, weights, intermediate data and outputs). (b) High bandwidth consumption of input vertex features. (c) Operations breakdown. Cora(CA), Citeseer(CS), and Pubmed(PB).

assignment table does not follow a sequential pattern, which leads to irregular memory access. Secondly, the randomness of data in assignments brings high irregularity to the second phase look-up operations for subvectors. The second phase look-up operations rely on the indexes from the first phase look-up operations that are random. Therefore, the random indexes and the irregular access to them introduce high irregularity to the look-up operations for subvectors, which makes the typical prefetching and caching techniques ineffective.

The unique computation pattern. The look-up chain operation introduces index calculations and look-up operations with specific dependencies. The second phase look-up operation to the centroids relies on the index from the first phase look-up operations. The reconstructed vectors are processed following the look-up chain. These chain-like operations require a specific datapath. In addition, look-up operations show significant operation redundancy. We profile the look-up operations for reconstructing the vectors, as shown in Fig. 3 (c). It shows that on average more than 71% of operations are performed to access the same centroids including the all-zero centroid and top-10% centroids. The all-zero centroids have no effect on the final results. The repeated operations on a small portion of top-10% centroids occupy a majority of non-sparse operations(25% in Pubmed). By reusing the results of repeated operations and eliminating unnecessary operations, it is possible for a performance improvement. Therefore, it motivates us to propose an efficient architecture to tackle the unique patterns.

The need for a specific accelerator and related work. The unique data access and computation patterns pose significant challenges to hardware. CPUs and GPUs struggle to manage the memoization results while the look-up chain heavily utilizes the memoization table. Although designing specialized accelerators is a popular paradigm, CNN accelerators designed to exploit the regular matrix multiplication in neural networks suffer from inefficiency for the irregular graph operators in GCN. To this end, specialized GCN accelerators [**?**], [2]–[6] have been proposed. They propose to support neural networks and sparse graph operators, and leverage the graph sparsity for performance optimization. However, they are unable to support the unique look-up chain operations. First, existing GCN accelerators lack control of the memoization table that the look-up chain relies on. Second, the look-up chain exhibits unique computation patterns and requires a special datapath. Especially, operation redundancy needs a specialized architecture to detect and reduce the types of redundant operations. This paper aims to propose an efficient accelerator that can not only leverage PQ distilling semantic redundancy to reduce bandwidth consumption but also reduce types of operation redundancy.

IV. Architecture Design

Next, we propose an efficient PQ-based GCN accelerator to support unique data access and computation patterns.

A. Workflow of GPACE

The workflow of GPACE is abstracted into three phases: look-up chain phase, vertex processing phase, and feature update phase, as described in Fig. 4 (a). ***(1) Look-up chain phase*** primarily performs the look-up chain operations that consist of two phases of look-up operations with dependencies. The vector features are reconstructed by looked-up subvectors for subsequent phases. ***(2) Vertex processing phase*** follows the look-up chain phase and performs local processing of the looked-up subvectors. The looked-up neighbor features are aggregated to the destination vertex. ***(3) Feature update phase*** transforms vertex features into a new dimension using learned weights, which exhibits obvious data reuse characteristics.

B. Architecture Overview

To efficiently support the aforementioned three phases, GPACE is implemented with specialized components, as shown in Fig. 4(b). The Edge Parser parses edge information from the edge buffer and forwards the parsed neighbor IDs to look-up engines for look-up operations. To support the look-up chain with unique data access and computation patterns, Look-up Engines receive the neighbor IDs and perform the look-up chain. The Codebook stores the sub-vectors of vertex features. To exploit the codebook parallelism, GPACE implements multiple parallel look-up engines, each of which is equipped with a Vertex Processing Unit to support vertex processing phase. The Vector Reduce Unit is designed to gather the intermediate aggregation results from parallel VPUs. To support the feature update phase with obvious data reuse characteristics, GPACE is designed with Vertex Updater. The Central Controller coordinates each unit and schedules the dynamical execution flow.

C. Micro-architecture

1) Look-up Engine: To support look-up chain operations with unique data access and computation patterns, we design look-up engines with dedicated components. As shown in Fig. 4(c), it consists of an Index Retriever, Centroid Retriever, Assignment Table, Codebook Table, and Sparsity Eliminator. As accessing the subvectors requires performing ***look-up chain*** operations with two-phase dependent operations, the look-up engine is implemented with a dedicated datapath. First, the neighbor IDs are parsed by the Edge Parser. A batch of neighbor IDs of the destination vertex is sent to the Index Retriever. The Index Retriever calculates the addresses according to the neighbor IDs and looks up the indexes stored in the assignment table. The looked-up indexes are subsequently forwarded to the vector selector to filter out unnecessary operations on all-zero vectors before looking up them. Then, the selected indexes are sent to the Centroid Retriever for second-phase look-up operations. According to each retrieved index, the Centroid Retriever performs the second-phase look-up operation to the subvector in the codebook table. When performing the look-up

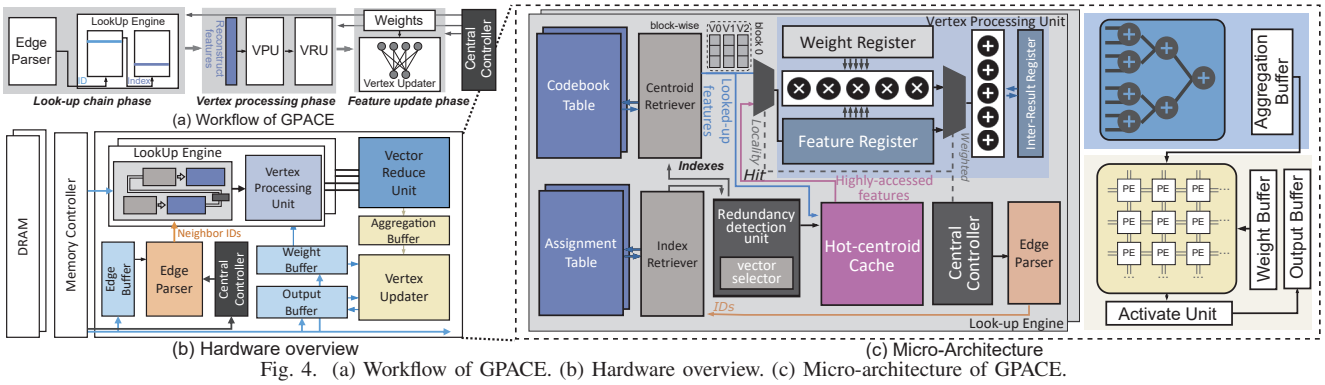

Fig. 4. (a) Workflow of GPACE. (b) Hardware overview. (c) Micro-architecture of GPACE.

operations, there are two design choices: block-wise and vertex-wise dataflow. To minimize the data transfer of intermediate results, GPACE implements a block-wise dataflow. The vertex-wise dataflow requires frequent intermediate data transfer of each block between on-chip and off-chip memory when the blocks are switched. While the block-wise approach processes the same feature block of vertices before switching to the next block. As a result, the intermediate results of each block are held on-chip until the final results are obtained, which significantly saves the off-chip memory access energy. We quantitatively compare the normalized off-chip memory access of the two choices, GPACE achieves 48.7× data access saving on average with block-wise dataflow than vertex-wise dataflow.

2) Vertex Processing Unit: The Vertex Processing Unit (VPU) is designed to efficiently support neighbor aggregation operations. Each look-up engine is equipped with a VPU to support the local processing of looked-up vectors. The VPU performs two crucial functions: local processing and hot-centroid cache management. First, one of the key functions is to perform the local processing of looked-up subvectors and neighbor aggregation. The features of neighbors retrieved from the codebook are stored in the feature register and are aggregated by VPU. For the weighted edge processing, VPU first performs the element-wise multiplication of features and weights. Second, the VPU incorporates hot-centroid cache management. Look-up operations to the highly-accessed centroid show high redundancy. To address this, the redundancy detection unit and hot centroid cache are implemented within each VPU to reduce the redundant address calculations and look-up operations. VPU is responsible to dynamically manage the hot-centroid cache.

3) Vector Reduce Unit: To gather the results from the look-up engines, GPACE implements a Vector Reduce Unit (VRU) at a high level. VRU essentially performs reduction operations of the intermediate results from multiple VPUs. For this purpose, this module is implemented using an adder tree with N adders to perform the fast reduction of N features in the log cycle. In this module, the final vertex features that aggregate the information of all neighbors are generated.

4) Vertex Updater: Vertex Updater is responsible for the efficient execution of the Vertex Update phase. The vertex feature length is transformed to a new dimension by learned weight, which is essentially a matrix-vector multiplication (MVM). As the computation in the Vertex Update phase has obvious data reuse characteristics, GPACE adopts a classic systolic array design in Vertex Updater to fully exploit the parallelism and data reuse, as shown in Fig. 4. The weight buffer is designed to exploit weight temporal locality. The outputs are forwarded to a one-dimensional activation unit for activation.

D. Redundancy-aware Look-up Policy

We conduct an in-depth analysis of the look-up chain and notice that there are two types of operation redundancy. **(1) locality-caused redundancy** refers to redundant operations to those frequently accessed centroids. **(2) sparsity-caused redundancy** refers to the operations to all-zero centroid. It requires identifying the all-zero centroid before looking up it. Therefore, in addition to leveraging PQ to distill the semantic redundancy, we propose a redundancy-aware look-up policy and implement GPACE in a redundancy-aware architecture to detect and eliminate the types of operation redundancy.

Optimization of locality-caused redundancy. The proposed optimization of locality-caused redundancy is based on the observation that the reconstruction of a common neighbor is performed repeatedly. As vertices in real-world graphs tend to cluster together and share many common neighbors, look-up operations to a small portion of vertices occur frequently. Therefore, the repeated look-up operations to these vertices are redundant. This is referred to as the locality of the graph, which motivates us to reuse the results of hot look-up operations. To achieve this, GPACE is implemented with a hot-centroid cache to improve the reuse of highly accessed vectors and allow direct access to them without repeating address calculations and look-up operations. As shown in Fig. 5, features of highly reusable neighbors are directly accessed from the hot-centroid cache if the neighbor ID hits the cache. The corresponding redundant calculations and operations are reduced. The cache and dynamic dataflow are managed by VPU. Specifically, in the first access to the highly reusable neighbor, the feature vectors are retrieved from the codebook table and meanwhile stored in the hot-centroid cache. The later repeated feature access will be reduced. The highly-accessed vertices can be identified through offline profiling according to the power-law distribution that a small number of vertices have a relatively high out-degree. We profile the IDs of the top-k out-degree vertices and dynamically store these profiled vertices in the hot-centroid cache.

Optimization of sparsity-caused redundancy. In addition, we notice that there is also sparsity-caused redundancy. The significant portion of redundant operations to all-zero vectors

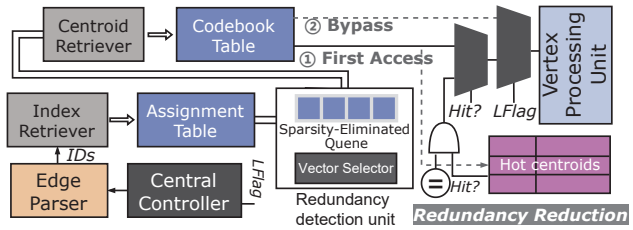

Fig. 5. Optimizations for locality-caused and sparsity-caused redundancy.

illustrated in Fig. 3(c) motivates us to propose a redundancy reduction mechanism. Many subvectors will be mapped to the same centroid with all-zero values due to the feature sparsity, which has no impact on the final results. Therefore, it is necessary to eliminate the sparsity-caused redundancy. It requires to identify the all-zero centroid before looking up to it. Fortunately, the index of the all-zero vector can be fixed and set to the first index in the assignment table during offline training of PQ-GCN. Therefore, GPACE is designed with a vector selector and sparsity-eliminated queue to detect and reduce the redundant operations on all-zero vectors before looking up them. Specifically, the vector selector filters out indexes to all-zero vectors and selects other indexes to the Sparsity-Eliminated Queue for later operations. If the index equals the first index, it indicates that the subvector to be looked up is an all-zero vector at the first index in centroid codebook. Otherwise, the index will be selected to the Sparsity-Eliminated Queue to achieve non-redundancy operations.

V. Evaluation

A. Experimental Setup.

Accelerator simulator. To evaluate the performance of GPACE, we simulate the micro-architectural behaviors of each module and build a cycle-accurate simulator, which is integrated with Ramulator [12] to simulate the behaviors of HBM.

ASIC synthesis. To measure the area and energy consumption of GPACE, we implement GPACE in Verilog and perform synthesis using Synopsys Design Compiler with the TSMC 12nm process and energy is estimated by [13]. The area of on-chip memory is calculated by Cacti [14].

Baselines and Datasets. To evaluate GPACE, we pick four graph datasets from the real world as listed in Table I. We compare GPACE with three different baselines, two general-proposed processors, and SOTA GCN accelerators on GCN model [1]. 1) CPU-PyG. CPU platform is equipped with Intel Xeon Platinum 8163@2.5GHZ and HBM@255GB/s to run the PQ-GCN. 2) GPU-PyG. GPU platform is equipped with NVIDIA Tesla P100 and HBM@732GB/s to run the PQ-GCN [10]. CPU and GPU platforms adopt SOTA framework Pytorch geometric(PyG). 3) HyGCN supports the hybrid execution and propose to reduce sparse edges. 4) ReGNN supports a redundancy-eliminated mechanism. For a fair comparison, baseline accelerators and GPACE are equipped with the same amount of resources and DRAM bandwidth. Table II shows detailed configurations of the baselines and GPACE.

TABLE I
Dataset Information.

Graph	#Vertices	#Edges	Feature Length	Feature Density
Cora(CA)	2708	10556	1433	1.27%
Citeseer(CS)	3327	9104	3703	0.85%
Pubmed(PB)	19717	88648	500	10.0%
Reddit(RD)	232965	114615892	602	51.6%

TABLE II
Configurations of Baseline Architecture and GPACE.

	HyGCN	ReGNN	GPACE
compute resources	1GHz @ 8 SIMD 16cores, 16×64 systolic array	1GHz @ 128 PEs, 16×64 systolic array	1GHz @ 32×4 PEs, 16×64 systolic array
On-chip memory	Same as the total set in GPACE	Same as the total set in GPACE	96KB(Assignment table), 32KB(Codebook Table), 128KB(Aggregate), 32KB(Weight), 32KB(Output), 16KB(Cache)
Off-chip memory	256GB/s HBM 1.0	256GB/s HBM 1.0	256GB/s HBM 1.0

B. Experimental Results.

Area consumption. GPACE shows a small area consumption and footprint of $2.8mm^2$. A large portion of the area is contributed to computation resources, which occupy 81.2% of the total area. The on-chip storage occupies 18.2% of the total area, which shows a small memory footprint. This is because GPACE achieves a small memory footprint and data size reduction through the efficient support of PQ. Compared with other GCN accelerators, GPACE consumes considerably fewer on-chip memory resources and occupies a smaller area.

Speedup. We compare the performance of GPACE with baselines, as shown in Fig. 6(a). GPACE achieves 390.4×, 56.3×, 8.6×, and 5.1× speedup on average compared with CPU-PyG, GPU-PyG, HyGCN, and ReGNN respectively. The performance improvement can be attributed to three key factors. Firstly, getting benefit from the proposed specialized architecture, GPACE performs efficient look-up chain operations and achieves a significant reduction of input-dominated bandwidth consumption. Secondly, GPACE carefully optimizes the dataflow and coordinates each component. Thirdly, with the proposed redundancy-aware look-up policy, GPACE effectively reduces types of redundant operations. Compared with CPU-PyG and GPU-PyG, GPACE achieves a performance improvement. While GPU offers abundant computational resources, it lacks efficient support for irregular look-up chain operations, resulting in inefficient memory access and resource underutilization. HyGCN and ReGNN have to tolerate a large amount of data movement and consume a large bandwidth. Moreover, HyGCN introduces redundant operations on the common neighbors. GPACE achieves performance gains through the proposed redundancy-aware look-up policy.

Energy consumption. Fig. 6 (b) shows the normalized energy consumption of GPACE and baselines. GPACE achieves on average 948.8×, 119.1×, 6.5×, and 4.1× energy savings compared with CPU-PyG, GPU-PyG, HyGCN, and ReGNN respectively. Energy savings mainly come from low latency, bandwidth saving, and redundant operations reduction. With the proposed specialized architecture, GPACE can efficiently support the unique look-up chain and leverage PQ to reduce bandwidth consumption and thus achieve efficient energy consumption. In addition to distilling the semantic redundancy by leveraging PQ, GPACE can reduce types of operation redundancy with proposed redundancy-aware look-up policy.

Accuracy. Applying PQ to GCN provides great opportunities to distill the semantic redundancy. The accuracy loss is limited to about 1% on average on all datasets which is negligible compared to its benefits.

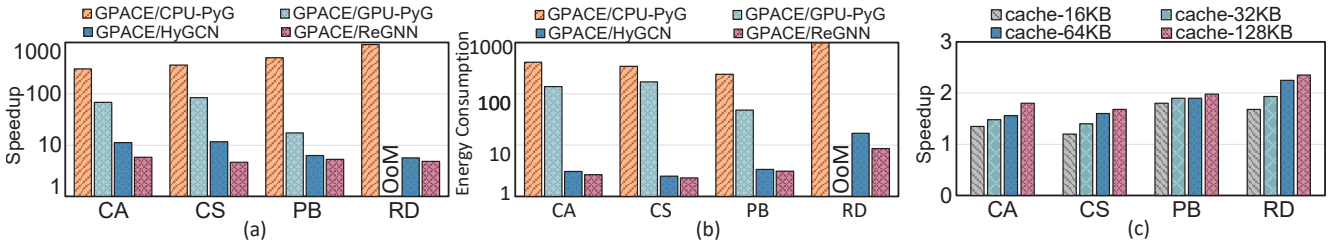

Fig. 6. (a)Performance comparison. (b)Energy consumption. OoM: Out of Memory. (c)Effects of hot centroid cache size.

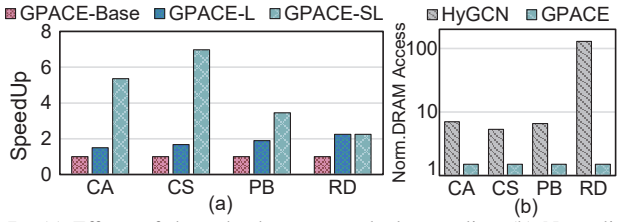

Fig. 7. (a) Effects of the redundancy-aware look-up policy. (b) Normalized DRAM Access reduction over HyGCN.

C. Analysis of Redundancy-aware Look-up Policy.

In this section, we analyze the impact of the proposed redundancy-aware look-up policy, which includes the elimination of locality-caused redundancy and sparsity-caused redundancy. As shown in Fig. 7(a), GPACE with the proposed redundancy-aware look-up policy can achieve 4.5× redundant operation reduction than GPACE without the policy.

To study the individual effects of reducing locality-caused redundancy and sparsity-caused redundancy, we evaluate GPACE under different configurations. GPACE-Base is implemented without the redundancy-aware look-up policy. GPACE-L is implemented with only the elimination of locality-caused redundancy. GPACE-SL is implemented with the elimination of both locality-caused and sparsity-caused redundancy. As shown in Fig. 7(a), GPACE-L achieves 1.9× performance improvement compared with GPACE-Base through locality-caused redundancy elimination. GPACE-SL achieves a further 2.4× performance improvement compared with GPACE-L through sparsity-caused redundancy elimination and achieves a total 4.5× performance improvement compared with GPACE-Base. It is shown that GPACE-L achieves higher performance improvements on Pubmed and Reddit than on Croa and Citeseer, which indicates that the locality-caused redundancy is more pronounced in Pubmed and Reddit. Therefore reducing locality-caused redundancy gains more benefit on these datasets. The comparisons of GPACE-SL and GPACE-L show that GPACE-SL achieves higher performance improvement on Croa and Citeseer than on Pubmed and Reddit. This is because vertex features in Croa and Citeseer are sparser(only 1.27% and 0.85% density) than Pubmed and Reddit(10.0% and 51.6%). Therefore, the proposed sparsity-caused redundancy elimination brings higher performance improvement on sparse datasets.

D. Memory Bandwidth Reduction.

We compare the bandwidth consumption by counting the DRAM Access of GPACE and HyGCN. As shown in Fig. 7(b), GPACE achieves 38× DRAM Access savings on average compared with HyGCN. GPACE is designed with a specific architecture that leverages PQ to distill semantic redundancy and reduce bandwidth consumption. Besides, the carefully optimized dataflow improves data reuse and reduces data access.

E. Analysis of the Hot Centroid Cache.

To investigate the impact of cache size on the performance of the proposed redundancy-aware look-up policy, we weep the size of the hot-centroid cache from 16 KB to 128 KB. As shown in Fig. 6(c), the proposed redundant-aware policy effectively utilizes the hot-centroid cache to reduce the redundant operations and achieves performance improvement. It is noticed that in RD, the large cache size yields higher performance improvements. This is because RD exhibits a higher density, which makes it show high data reusability and can leverage large size of hot-centroid cache to reduce redundant operations.

VI. Conclusion

In this paper, we propose GPACE, an energy-efficient GCN accelerator that can fully harness the potentials of PQ to reduce the bandwidth consumption and memory footprint. GPACE is designed in a lookup-efficient architecture to support the unique data access and computation pattern of PQ-GCN. In addition to leveraging PQ to distill semantic redundancy, we implement a redundancy-aware architecture. The evaluation results demonstrate GPACE achieves outstanding energy efficiency.

VII. Acknowledgement

This paper is supported by the National Natural Science Foundation of China(NSFC) under grant No. 62222411 and No. 62202453, China Postdoctoral Science Foundation under grant 2022M713207, Zhejiang Lab under Grants 2021PC0AC0. The corresponding authors are Ying Wang and Shengwen Liang.

References

[1] Thomas N Kipf et al. Semi-supervised classification with graph convolutional networks. *ICLR*, 2017.
[2] Mingyu Yan et al. Hygcn: A gcn accelerator with hybrid architecture. In *HPCA*. IEEE, 2020.
[3] Cen Chen et al. Regnn: A redundancy-eliminated graph neural networks accelerator. In *HPCA*. IEEE, 2022.
[4] Gopikrishnan Raveendran Nair et al. Fpga acceleration of gcn in light of the symmetry of graph adjacency matrix. In *DATE*, 2023.
[5] Jacob R. Stevens et al. Gnnerator: A hardware/software framework for accelerating graph neural networks. In *2021 DAC*, pages 955–960, 2021.
[6] Ogbogu et al. Accelerating graph neural network training on reram-based pim architectures via graph and model pruning. *IEEE TCAD*, 2023.
[7] Junwhan Ahn et al. A scalable processing-in-memory accelerator for parallel graph processing. In *ISCA*, 2015.
[8] and others Hamilton. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.
[9] Nagasaka et al. Batched sparse matrix multiplication for accelerating graph convolutional networks. In *CCGRID*, pages 231–240. IEEE, 2019.
[10] Linyong Huang et al. Epquant: A graph neural network compression approach based on product quantization. *Neurocomputing*, 2022.
[11] Paul-Edouard Sarlin et al. Superglue: Learning feature matching with graph neural networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020.
[12] Yoongu Kim et al. Ramulator: A fast and extensible dram simulator. *IEEE Computer architecture letters*, 2015.
[13] Song Han et al. Eie: Efficient inference engine on compressed deep neural network. In *ISCA*, 2016.
[14] Cacti. In *[Online]. Available:http://www.hpl.hp.com*.