EGMA: Enhancing Data Reuse and Workload Balancing in Message Passing GNN Acceleration via Gram Matrix Optimization

Fangzhou Ye*, Lingxiang Yin*, Amir Ghazizadeh Ahsaei, Hao Zheng
Department of Electrical and Computer Engineering, University of Central Florida, Orlando, FL, USA
{fa011718, lingxiang.yin, amir.g, hao.zheng}@ucf.edu

ABSTRACT

Graph Neural Networks (GNNs) have been widely used to handle intricate graph-related problems, in which complex vertex and edge operations are performed in the form of message passing between vertices. Such complex GNN operations are highly dependent on the graph structure and can no longer be characterized as sparse-dense or general matrix multiplications. Consequently, current matrix-based data reuse and workload balancing optimizations have limited applicability to Message Passing-based GNN acceleration. In this paper, we leverage the mathematical insights from Gram Matrix to simultaneously exploit data reuse and workload balancing opportunities for message passing-based GNN accelerations. Upon this insight, we further propose a novel accelerator, named EGMA, that can efficiently facilitate a wide range of GNN models with improved data reuse and workload balance. Consequently, EGMA can achieve performance speedup by 1.57×, 1.72×, and 1.43× and energy reduction by 38.19%, 34.02%, and 24.54% on average compared to Betty, FlowGNN, and ReGNN, respectively.

ACM Reference Format:
Fangzhou Ye*, Lingxiang Yin*, Amir Ghazizadeh Ahsaei, Hao Zheng. 2024. EGMA: Enhancing Data Reuse and Workload Balancing in Message Passing GNN Acceleration via Gram Matrix Optimization . In *61st ACM/IEEE Design Automation Conference (DAC '24), June 23–27, 2024, San Francisco, CA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3649329.3655962

1 INTRODUCTION

Graph neural networks (GNNs) have demonstrated exceptional capabilities in a wide range of graph-related tasks, including but not limited to computer vision, social networks, natural language processing(NLP), and recommendation systems [1–4], among others. Conventional GNNs typically utilize a message passing technique, in which messages are exchanged between vertices followed by neural networks. The message passing process therefore is highly dependent on graph structure. However, real-world graphs exhibit significant variations in vertex connectivity [5], leading to high sparsity and irregularity. This poses serious challenges to graph data reuse and workload balance.

*Equal Contribution.

DAC '24, June 23–27, 2024, San Francisco, CA, USA

ACM ISBN 979-8-4007-0601-1/24/06…$15.00
https://doi.org/10.1145/3649329.3655962

Table 1: The comparison of EGMA with other state-of-the-art accelerators.

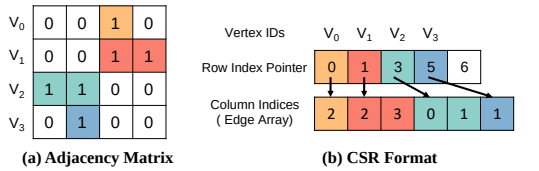
Accelerator	Message Passing	Data Reuse	Unifed PE	Worload Balance	
				Aggr.	Update
AWB-GCN [6]	✗	✗	✗	✓*	✓*
GCNAX [7]	✗	✓*	✓*	✗	✗
ReGNN [8]	✓	✓	✗	✗	✓
FlowGNN [9]	✓	✗	✗	✗	✓
EGMA	✓	✓	✓	✓	✓

∗: optimized for sparse-dense matrix multiplications.

Even though specialized graph accelerators [6–10] have been proposed to address the data reuse and workload balance issues, they are typically optimized for sparse-dense matrix multiplications [6, 11]. For example, GCNAX [7] exploited the use of loop fusion and reordering to unify dense and sparse matrix multiplications and reduce costly DRAM access. AWB-GCN [6] proposed a runtime workload distribution technique that can adjust workload allocation depending on runtime resource utilization. I-GCN [12] employed a breadth-first search approach to identify dense graph connectivity, thus making it suitable for dense matrix multiplication with improved workload balance. Unfortunately, all of the mentioned prior works have limited applicability to message passing GNN models [13, 14].

To support the message passing mechanism, FlowGNN [9] introduced a dataflow architecture to pipeline message aggregation and vertex update with disjoint computation units. These computation units are connected by a complex interconnection network to withstand erratic communication. Similarly, ReGNN [8] utilized a redundancy-eliminated neighborhood message passing to eliminate redundant aggregation computations. However, its parallelism is also restricted by separate graphs and neural operations. More importantly, both workload balance and graph data reuse remain unexplored in different GNN execution phases.

To address the mentioned challenges, we proposed an accelerator architecture, called EGMA, that can dynamically discover graph data reuse opportunities and achieve both vertex and degree balancing. Specifically, EGMA is built upon Gram matrix - the product of the adjacency matrix and its transpose ($A \cdot A^T$). This matrix mathematically provides data reuse (mutually-shared neighboring nodes between a pair of vertices), vertex, and edge information. Upon this new insight, we further developed a Gram matrix-based scheduling that leverages such information to optimize graph data reuse while preserving workload balance. Following that, we further propose a novel accelerator architecture to dynamically support the proposed Gram matrix-based scheduling. In addition, the proposed accelerator employs a unified architecture to pursue fine-grained parallelism for a variety of GNN models. The main contributions of this paper are:



(a) Adjacency Matrix (b) CSR Format

Figure 1: (a) An example of Adjacency Matrix, and (b) its CSR Format.

- **A Novel Gram Matrix-based Scheduling for Dynamic Graph Data Reuse and Workload Balancing**: We proposed a novel algorithm based on Gram Matrix to exploit dynamic graph data reuse. In addition, the proposed algorithm can simultaneously balance both edge and vertex workloads for distinct GNN execution phases.
- **A Unified Accelerator Architecture for Message Passing GNN models**: We designed a unified accelerator architecture, consisting of a Gram matrix-based scheduler and a unified PE architecture. The proposed scheduler can dynamically process graph metadata to obtain the reuse, edge, and vertex information for workload scheduling. Furthermore, the unified PE architecture can allow the pursuit of fine-grained parallelism (i.e., edge and vertex parallelism) as compared to counterparts (i.e., pipeline parallelism).

We evaluate the proposed EGMA, and the evaluation result shows that the proposed EGMA design can achieve 1.43× speedup and 24.54% energy reduction on average over the state-of-the-art GNN accelerators [8].

2 BACKGROUND AND MOTIVATION

2.1 CSR Format Matrix

Given the sparsity of real-world graphs, significant research [15, 16] has proposed a collection of compression techniques to eliminate zero elements, reducing storage requirements. Despite many compression choices, graph data is often encoded using the Compressed Sparse Row(CSR) format [17–19], which condenses edge information into arrays, making its space proportional to the number of vertices and edges with adequate metadata overheads.

For example, the adjacency matrix is used to represent graph connectivity, in which each non-zero element indicates an edge connectivity between two vertices. Moreover, the vertex *degree* information, denoted as d_u for a node u, can be estimated by counting non-zero elements in each row of the adjacency matrix:

$$d_u = \sum_v A[u, v], \forall v \in V \tag{1}$$

Where A is the adjacency matrix and $u \in V$ is a vertex.

To reduce the storage requirement for an adjacency matrix, the non-zero elements can be eliminated in the CSR format. As shown in Fig. 1, the first row of the adjacency matrix can be represented in the CSR format by using row index and edge arrays.

2.2 Message passing-based Graph Neural Networks

GNNs commonly employ a message passing approach [13] to exchange vertex and edge features, followed by neural network operations. Typically, the message passing can be abstracted as two

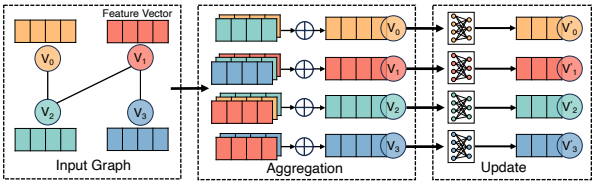


Figure 2: Message passing mechanism in GNNs. Generic GNN execution phases: message aggregation and vertex update.

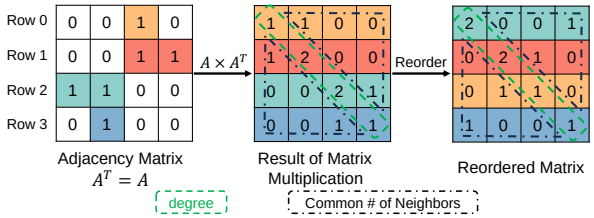


Figure 3: An example of Gram Matrix to obtain graph reuse, vertex, and edge information.

execution phases - message aggregation and vertex update. For an input graph (V, E) where V is the vertex and E is the edge, each vertex gathers information from its neighbors and updates the received message information into its own features. As shown in Fig. 2, V_2 has two neighboring vertices V_0 *and* V_1. In the pull-based model [20], "message" will be transferred from source vertex V_0 *and* V_1 to target vertex V_2. The definition of the message passing mechanism is formally presented as follows:

$$\mathcal{M}^{(k)}_{\mathcal{N}(u)} = AGGREGATE(h^{(k)}_v, \ \forall v \in \mathcal{N}(v)) \tag{2}$$

Where the *Aggregate* function aggregates multiple feature vectors from *n's* neighbors ($\mathcal{N}(v)$) *hidden embeddings* $h^{(k)}_v$ correspond to each node $v \in \mathcal{N}(v)$ to "message" $\mathcal{M}^{(k)}_{\mathcal{N}(u)}$. As shown in Fig. 2, the vertex update in each GNN layer can be formulated as:

$$h^{(k+1)}_u = UPDATE^{(k)}(h^{(k)}_u, \mathcal{M}^{(k)}_{\mathcal{N}(u)}) \tag{3}$$

Here, $\mathcal{M}^{(k)}_{\mathcal{N}(u)}$ is based on its aggregated neighborhood information. The *Update* function, merges $\mathcal{M}^{(k)}_{\mathcal{N}(u)}$ with the prior embedding $h^{(k)}_u$ of node u to produce the refreshed embedding $h^{(k+1)}_u$. The initial embeddings at $k = 0$ are initialized with the input features for all nodes, i.e., $h^{(0)}_u = X_u, \ \forall u \in V$. After executing $k + 1$ iterations of the GNN message passing, we can utilize the output from the final layer to establish the embeddings for each node.

2.3 Motivation

In GNN computations, the redundancy in computation and communication stems from the scenario where multiple vertices have shared neighbors. Mathematically, finding shared neighbors between a pair of vertices can be expressed as the product of two rows in the adjacency matrix:

$$\begin{aligned} Common\_neighbors &= \mathcal{N}(u) \cap \mathcal{N}(w), \ u, \ w \in V \\ &= A[u, V] \times A[w, V]^T, \ u \neq w \end{aligned} \tag{4}$$

Where we use *Common_neighbors* to denote the value quantifying the relationship between nodes u and w.

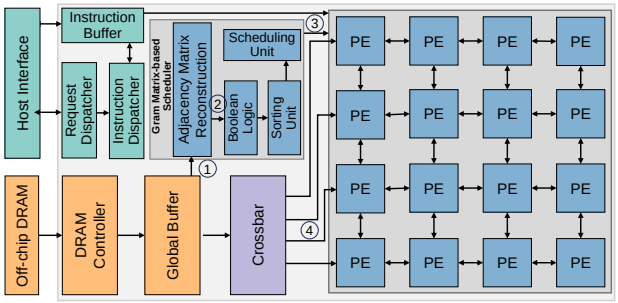


Figure 4: Overview of the Proposed EGMA Accelerator Architecture (a 4×4 PE array example).

Generalizing the mathematical expression to a graph, the process involves the multiplication of the adjacency matrix and its transpose. In graph theory, such a matrix multiplication is called Gram Matrix. As illustrated in Fig. 3, the "Gram matrix" can provide a global view of commonly shared neighbors of all the vertices, connectivity patterns, and node similarity. For example, the diagonal elements of the Gram matrix represent the degree of each vertex, while the upper and lower diagonals indicate mutual vertices that are shared between vertex pairs. In this paper, we will use Gram Matrix to explore graph data reuse and workload balancing opportunities.

3 PROPOSED EGMA DESIGN

In this paper, we propose a unified accelerator architecture termed EGMA to improve data reuse, workload balancing, and parallelism for message passing GNN models. Specifically, EGMA has two unique designs: a Gram-matrix-based scheduler and a unified PE design. The proposed scheduler can identify commonly shared neighbors between vertices while balancing edge and vertex workloads. Moreover, the proposed unified accelerator architecture allows the pursuit of fine-grained parallelism (e.g., edge and vertex).

3.1 Proposed Accelerator Architecture

The detailed EGMA architecture is depicted in Fig. 4. As mentioned earlier, prior work employs disjoint computation units to pipeline GNN execution phases. To pursue fine-grained parallelism, a unified architecture is proposed to support both aggregation and vertex update. As such, this can eliminate the need for additional communication between separate units. The proposed accelerator includes a DRAM controller, a global buffer (GLB), a Gram matrix-based scheduler, an instruction buffer, a request dispatcher, an instruction dispatcher, and a processing element (PE) array.

The control unit receives requests from the host and stores them in the request dispatcher. The request dispatcher sends the compiled request to the instruction buffers. The instruction dispatcher keeps track of instruction issuing and completion. The controller generates addresses for the instruction buffer, which forwards instructions to both the PE array and scheduler. The DRAM is connected to the proposed accelerator through the DRAM controller. To increase the bandwidth between the GLB and PE array, a crossbar is implemented to support all-to-all communication. The PE array consists of multiple PEs interconnected by a mesh topology.

3.1.1 Proposed PE Architecture. The proposed PE architecture is shown in Fig. 5, which consists of feature/output/weight buffers, a

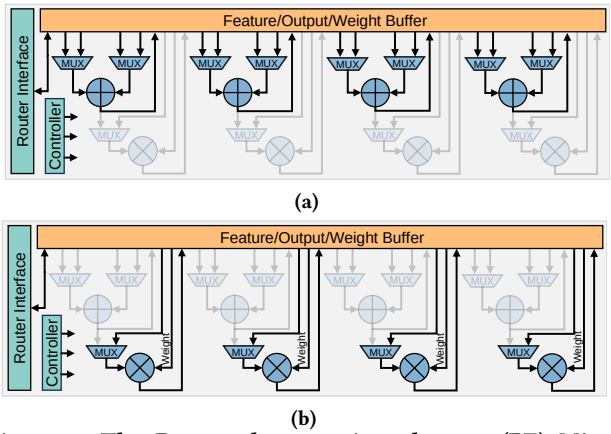


Figure 5: The Proposed processing element (PE) Microarchitecture: (a) Aggregation Phase (b) Update Phase.

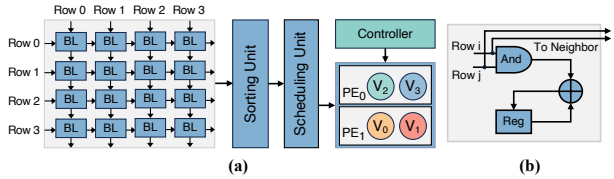


Figure 6: (a) Proposed Gram Matrix-based Scheduler Architecture, and (b) Proposed Boolean Gates for Adjacency Matrix Multiplication

controller, a router interface, and a flexible multiply–accumulation (MAC) array. To support the distinct computation characteristics of GNN execution phases, the MAC array could be reconfigured to support feature aggregation and matrix multiplications. For example, Fig. 5 (a) shows the aggregation of collected features from neighbor nodes in the highlighted compute unit. Similarly, the MAC array can be configured to support the vertex update, in which the aggregated feature vector will be multiplied by the weight matrix. The dataflow will be discussed in Section 3.3.

3.2 Proposed Gram Matrix-based Scheduler

Both edge and vertex variations will influence the performance of GNN execution. Specifically, the aggregation phase's workload is tied to the degree of the target vertices, as the number of neighbors corresponds to the vertex's degree. Conversely, the update phase's workload is determined by the total count of target vertices. Unfortunately, prior work has not simultaneously addressed both vertex and edge balancing issues. For example, conventional GCN accelerators optimized workload balancing in the form of dense matrix multiplication. On the other hand, graph processing accelerators are typically centered on edge balancing. To this end, we address this issue using the proposed Gram matrix-based scheduler.

3.2.1 Proposed Dynamic Gram Matrix Calculation. As the proposed EGMA accelerator stores the graph data in CSR format, the adjacency matrix will be reconstructed dynamically based on CSR metadata. The constructed adjacency matrix will be further sent to a simple Boolean logic array to calculate the Gram matrix.

As shown in Fig. 6 (a), the proposed Boolean Logic Array is employed to compute the product of adjacency matrix - $A \times A^T$. Due to the symmetric nature of the adjacency matrix, the transpose

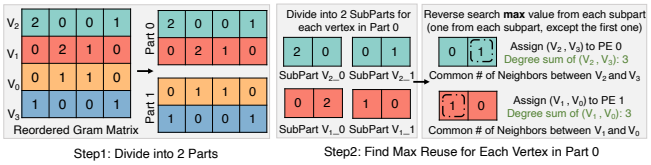


Figure 7: An example of the proposed algorithm assigning four vertices to two PEs.

of the adjacency matrix is identical to its original matrix. Therefore, each row or column of the adjacency matrix could be the input of the Boolean logic array. Each Boolean Logic (BL) element receives two rows of the adjacency matrix as input for both the *And* and *ADD Gates* as shown in Fig. 6 (b). It's worth noting that because the elements in the adjacency matrix are binary, the use of an *And Gate* is sufficient, thus reducing circuit complexity. The resultant values are then stored in a register within each BL element. In other words, the proposed Boolean Logic Array is a simplified version of a systolic array architecture specialized for Gram matrix computation. Similarly, if the adjacency matrix dimension exceeds the size of the proposed Boolean Logic Array, the adjacency matrix could be tiled and processed sequentially.

3.2.2 Gram Matrix-based Scheduling. After obtaining the Gram matrix, we propose a scheduling policy to leverage the global data reuse and degree information presented in each row and column. The proposed scheduling policy aims to achieve maximum vertex reuse while ensuring a balanced vertex and edge workload. The key idea is to rotate the rows and columns in the adjacency matrix: (1) group vertices with near-optimal shared neighbors and (2) ensure similar edge and vertex quantity among groups.

Degree-Aware Gram Matrix reordering: We perform a sorting algorithm to reorder the rows and columns of Gram matrix based on the degree information. After that, the degree of vertices will be ranked in descending order. The reordered diagonal degree information and ranked row index number are stored in a list. To reduce the overhead, we use an 8-bit precision for degree information, and the list allows 2048 vertices to be recorded. Given the power-law distribution, 8-bit precision is sufficient to capture the degree variation in real-world graphs.

Proposed Workload Scheduling: Based on the reordered Gram matrix, we need to find M groups of vertices, where M denotes the number of PEs. The proposed algorithm aims to create balanced workloads based on the vertices' degrees and their mutually shared nodes. To construct the groups, the proposed algorithm first divides the reordered Gram matrix into M groups. Subsequently, the algorithm assigns K (e.g., $K = \lceil N/M \rceil$) vertices to each group (G_i). As a result, each PE will host K vertices. The detail is shown in Algo. 1.

Specifically, the proposed algorithm starts processing the first M rows of the reordered Gram matrix iteratively in Algo. 1. Each row will be evenly divided into K parts, each with M vertices as shown in Algo. 1 (line 6-10). Within each part, we use the reverse search method (starting from the last column) to find the largest value which indicates the largest number of shared neighboring vertices. All the selected vertices will be added to the same group, which will be assigned to the same PE. This continues until all vertices have been assigned to PEs.

For example, as illustrated in Fig. 7, when a 4×4 reordered gram matrix is obtained, it is partitioned into two groups. We only need

Algorithm 1: Gram Matrix-based Workload Creation

```
1  Inputs: reor_mat : N × N Reordered Gram Matrix;
           M : Number of PEs; Vertex : List of all vertices
2  Outputs: G : Vertex Groups for PEs
3  for i ← 0 to M-1 do
4      temparray ← reor_mat[i]
5      Append Vertex[i] to G[i]
6      K ← ⌈N/M⌉
7      for j ← M to N-1; j ← j + M do
8          for k ← 0 to K − 1 do
9              subarrays[k] ← temparray[j : j + M]
10         end
11     end
12     for subarray in subarrays do
13         if all(Vertex) are not marked then
14             // Reverse search the first max value's index
15             Id ← index(max(subarray[:: −1]))
16             Append Vertex[Id] to G[i]
17             Mark Vertex[Id] as assigned
18         end
19     end
20 end
```

to process the first two rows of the adjacency matrix. For row 1, we find the last column V_3 that has the largest value. As such, V_2 and V_3 will be grouped and assigned to PE_0. Similarly, for row 2, V_0 has the largest value, which will be grouped with V_1. The second group will be assigned to PE_1.

3.3 Proposed Dataflow and Mapping

Both dataflow and mapping choices affect performance by exploiting temporal and spatial data locality via loop interchanging and spatial parallelism [6, 21]. In the process of message passing, GNN computations cannot be easily expressed as matrix multiplications. Also, the weight matrix is of a relatively lower dimensionality compared to CNNs [22]. This may limit the potential performance gains that can be achieved through loop interchanging for dataflow optimization.

We evenly distribute vertices and their feature vectors to the PE array, therefore, it avoids data duplication for feature vectors. At the message aggregation phase, the pull model is used which involves irregular communication patterns between PEs. At the update phase, the outer product has been chosen to avoid inter-PE communication. This is because the message aggregation already consumes a significant amount of inter-PE bandwidth. However, the outer product would require additional buffers at each PE to store the intermediate data. The weight matrix is delivered to the PE array in a broadcast manner.

4 EXPERIMENT AND EVALUATION RESULTS

4.1 Experiment Setup

Accelerator Simulator: We created a cycle-accurate simulator to evaluate the EGMA accelerator's performance and energy efficiency. This simulation faithfully reproduces every module within the

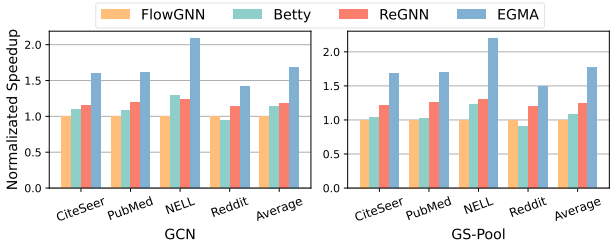


Figure 8: Normalized performance speedup of FlowGNN, Betty, ReGNN, and EGMA, with PE size of 6×6.

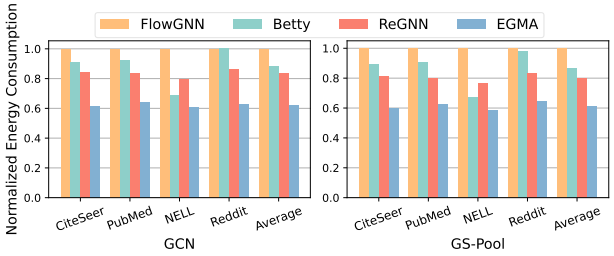


Figure 9: Normalized energy consumption of FlowGNN, Betty, ReGNN, and EGMA, with PE size of 6×6.

Table 2: Dimension, Density, and reuse percentage with different accelerator of the Graph Datasets.

Datasets	Dimension			Density	Reuse(PE size 64)			
	Vertex	Edge	Feature	A	Betty	FlowGNN	ReGNN	EGMA
CiteSeer	3327	9104	3703	0.11%	0.0%	0.0%	34.40%	30.09%
PubMed	19717	88648	500	0.023%	0.0%	0.0%	31.05%	25.35%
NELL	65755	251550	61278	0.0073%	0.0%	0.0%	12.82%	19.17%
Reddit	232965	114615892	602	0.21%	0.0%	0.0%	7.05%	6.67%

EGMA accelerator, and we have validated the timing behaviors of these modules by comparing them to the synthesized RTL design. The PE design employs a 1×4 MAC array using double-precision floating point adders and multipliers in each PE. The feature/weight/output buffers are sized according to graph size and sparsity. For example, the feature buffer is used to store the feature vectors, the size of the feature buffer should be large enough to hold all data. Due to the density of A does not exceed 0.21%, the feature size will not exceed 512KB, so we choose a 1MB feature buffer to cover the overhead. Similarly, we opt for a total of 2MB for weight buffers and 4MB for output buffers.

Baselines: We conduct a comparison between the performance and energy efficiency of EGMA and three distinct baseline designs, which are Betty [23], FlowGNN [9], and ReGNN [8]. For a fair comparison, all the baseline accelerators are scaled to have the same number of multiply-and-accumulate (MAC) units as EGMA.

GNN Models and Datasets: To benchmark the performance of EGMA accelerators, we implemented a set of typical GNN models on several traditional datasets as shown in Table 2. We employed two well-established GNN models, specifically the Graph Convolutional Network (GCN) [13, 24]and GraphSage-Pool (GS-Pool) [21]. GS-Pool is a classic spatial domain-based algorithm, which improves the traditional GCN and is widely utilized in semi-supervised classification tasks.

Energy and Area Evaluation: To assess the power and area of EGMA as an Application-Specific Integrated Circuit (ASIC) accelerator, we employed Synopsys Design Compiler along with the TSMC 40nm standard library for synthesis and waveform activity file generation. Subsequently, we utilized Synopsys PrimeTime PX, incorporating the waveform activity file to quantify both dynamic and static power consumption of EGMA. As for the on-chip buffer area and power, we utilized Cacti 6.0 [25] with 40nm technology.

4.2 Experiment Result

Performance Speedup: We compare the performance of EGMA with other baseline accelerators that are illustrated in Fig. 8. In this figure, we set the PE sizes to 36. The average performance speedup of the proposed EGMA over Betty, FlowGNN, and ReGNN are 1.57×, 1.72×, and 1.43× on all the datasets respectively. The performance gain stems from both data reuse and workload balance. First, in contrast to Betty and FlowGNN, EGMA performs better by incorporating data reuse exploitation. Second, while both ReGNN and EGMA consider data reuse, EGMA outperforms ReGNN due to its superior workload balance, leading to more efficient overall performance. Consequently, data reuse and workload balance are all essential in GNN execution. We will study further the scalability of EGMA in the following section.

Energy Consumption: We continue to study the energy consumption of EGMA with other algorithms, which are illustrated in Fig. 9. Across all GNN models, EGMA has the energy reduction by 38.19%, 34.02%, and 24.54% compared to FlowGNN, Betty, and ReGNN, respectively. The energy reduction comes from the reduced execution time and communication between distributed on-chip memory due to the increased data reuse.

Data Reuse Analysis: The detailed data reuse is analyzed in Table 2. Unlike Betty and FlowGNN which do not include data reuse optimization, ReGNN employs a dynamic redundancy-eliminated neighborhood message passing algorithm to enhance data reuse, yielding an average reuse of 21.33%. Compared to different accelerator baselines, EGMA demonstrates an average reuse percentage of 20.32%. These variations arise from our objective to maximize data reuse while ensuring workload balance. Throughout this process, there is an inherent trade-off, necessitating the sacrifice of some data reuse to maintain workload balance.

4.3 Workload Balance Analysis

Our novel accelerator design is to find workload balance at both GNN execution phases, which requires each PE to be assigned a similar amount of edges and vertices. We utilize the Interquartile Range (IQR), a statistical measure of dispersion in descriptive statistics, to depict the distribution of edges. We document the count of edges within each PE, as our proposed approach has ensured that all PEs have an equivalent number of edges. Since FlowGNN and ReGNN share identical vertex distributions in the PE array, we only compare FlowGNN. Given the limited space, we only present the comparison in CiteSeer Data with varying PE sizes.

In Fig. 10, it is evident that EGMA outperforms other baseline accelerators. As an illustration, with a PE size of 4, the Interquartile Range (IQR) for EGMA is 6, whereas, for Betty and FlowGNN, the IQR values are 990 and 74, respectively. This indicates that our approach could achieve a better workload balance, which could lead to higher resource utilization. On the other hand, please note that baseline accelerators only focus on equalizing vertex distribution, leading to an unbalanced edge distribution.

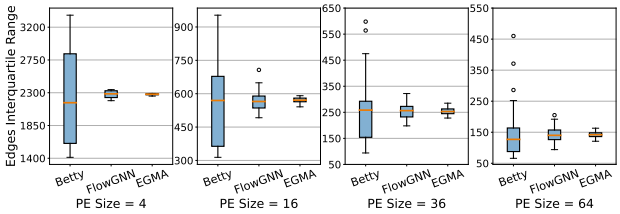


Figure 10: The workload evaluation: The Edge Interquartile range of Betty, FlowGNN, and EGMA with varying PE Size on CiteSeer dataset.

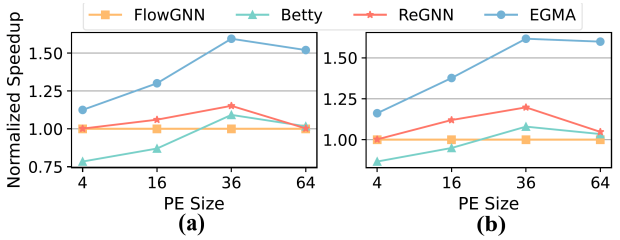


Figure 11: Scalability Evaluation: performance speedup of FlowGNN, Betty, ReGNN, and EGMA, with varying PE numbers on (a) CiteSeer dataset and (b) PubMed dataset.

4.4 Scalability Study

To evaluate the scalability of EGMA, we run GCN inference on the two datasets and vary the number of PEs. Fig. 11 presents that the correlation between performance and PE Size, in which the overall speedup scales well for EGMA compared to FlowGNN, Betty, and ReGNN. Betty utilizes the metis [26] graph partitioning method that optimizes the data structure and distribution of computational tasks to reduce communications. As the PE size increases, communication plays an important role in the performance. Hence, Betty demonstrates a better performance than FlowGNN. EGMA performs well due to its fine-grained parallelism.

4.5 Energy and Area Analysis

Energy breakdown: In Fig. 12(a), the breakdown of energy consumed by each operation involving Task Scheduling, Computation, Decoding, and *Read&Write* of EGMA are 19.91%, 0.77%, 28.78%, and 50.54% on average, respectively. *Read&Write* involves moving the data from the global buffer to the distributed local buffer, while decoding entails reading the indirection tables of the CSR compression format stored in the on-chip buffers. Meanwhile, computation involves performing MAC operations. Task scheduling encompasses the reordering of the gram matrix and the assignment of vertices to PE.

Area breakdown: Fig. 12(b) presents the distribution of area among EGMA's logical resources. Global buffer and local buffers collectively contribute to over 77.3% of the total area, whereas MAC units consume 14.6%. The mesh topology of EGMA results in an area of approximately 5.2%. Finally, the control logic takes 2.9% of the total area, which includes the control instructions in PE.

5 CONCLUSION

In this paper, we proposed a unified accelerator, based on Gram Matrix, to simultaneously exploit data reuse and workload balancing opportunities for massage passing-based GNN accelerations. To be

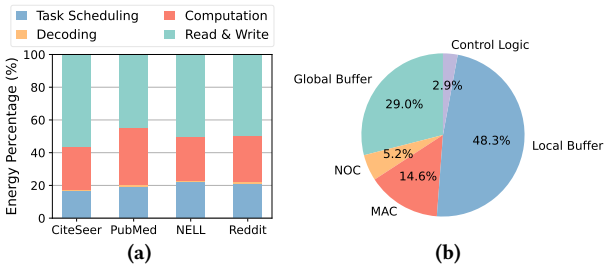


Figure 12: (a) Energy and (b) Area Breakdown.

specific, we propose a novel accelerator, named EGMA, that can efficiently facilitate a wide range of GNN models with improved data reuse and workload balance. Consequently, EGMA can achieve a performance speedup of 1.57×, 1.72×, and 1.43× and an energy reduction of 38.19%, 34.02%, and 24.54% on average compared to Betty, FlowGNN, and ReGNN, respectively.

REFERENCES

[1] Zonghan Wu et al. A comprehensive survey on graph neural networks. In *TNNLS*, pages 4–24, 2020.
[2] Fragkiskos D Malliaros et al. Clustering and community detection in directed networks: A survey. *Phys. Rep*, pages 95–142, 2013.
[3] Lingxiang Yin et al. Aries: Accelerating distributed training in chiplet-based systems via flexible interconnects. In *Proc. of ICCAD*, pages 1–9, 2023.
[4] Ziwei Zhang et al. Deep learning on graphs: A survey. In *TKDE*, pages 249–270, 2020.
[5] Lingxiao Ma et al. NeuGraph: Parallel deep neural network computation on large graphs. In *Proc. of USENIX ATC*, pages 443–458, 2019.
[6] Tong Geng et al. AWB-GCN: A graph convolutional network accelerator with runtime workload rebalancing. In *Proc. of MICRO*, pages 922–936, 2020.
[7] Jiajun Li et al. GCNAX: A flexible and energy-efficient accelerator for graph convolutional neural networks. In *Proc. of HPCA*, pages 775–788, 2021.
[8] Cen Chen et al. ReGNN: A redundancy-eliminated graph neural networks accelerator. In *Proc. of HPCA*, pages 429–443, 2022.
[9] Rishov Sarkar et al. FlowGNN: A dataflow architecture for real-time workload-agnostic graph neural network inference. In *Proc. of HPCA*, pages 1099–1112, 2023.
[10] Shengwen Liang et al. EnGN: A high-throughput and energy-efficient accelerator for large graph neural networks. In *TC*, pages 1511–1525, 2020.
[11] Guyue Huang et al. GE-SpMM: General-purpose sparse matrix-matrix multiplication on GPUs for graph neural networks. In *Proc. of SC*, pages 1–12, 2020.
[12] Tong Geng et al. I-GCN: A graph convolutional network accelerator with runtime locality enhancement through islandization. In *Proc. of MICRO*, 2021.
[13] Lingxiang Yin et al. Exploring architecture, dataflow, and sparsity for gcn accelerators: A holistic framework. In *Proc. of GLSVLSI*, page 489–495, 2023.
[14] Justin Gilmer et al. Neural message passing for quantum chemistry. In *Proc. of ICML*, pages 1263–1272, 2017.
[15] Se Jung Kwon et al. Structured compression by weight encryption for unstructured pruning and quantization. In *Proc. of the CVPR*, pages 1909–1918, 2020.
[16] Eric Qin et al. SIGMA: A sparse and irregular GEMM accelerator with flexible interconnects for DNN training. In *Proc. of HPCA*, pages 58–70, 2020.
[17] Pengcheng Yao et al. An efficient graph accelerator with parallel data conflict management. In *Proc. of PACT*, pages 1–12, 2018.
[18] Sanjay Gandham et al. Saga: Sparsity-agnostic graph convolutional network acceleration with near-optimal workload balance. In *Proc. of ICCAD*, pages 1–9, 2023.
[19] Eric Qin et al. Extending sparse tensor accelerators to support multiple compression formats. In *Proc. of IPDPS*, pages 1014–1024, 2021.
[20] Mingyu. Yan et al. HyGCN: A GCN accelerator with hybrid architecture. In *Proc. of HPCA*, pages 15–29, 2020.
[21] Will Hamilton et al. Inductive representation learning on large graphs. In *Proc. of NIPS*, pages 1025–1035, 2017.
[22] Alex Krizhevsky et al. Imagenet classification with deep convolutional neural networks. In *Proc. of NIPS*, page 1097–1105, 2012.
[23] Shuangyan Yang et al. Betty: Enabling large-scale GNN training with batch-level graph partitioning. In *Proc. of ASPLOS*, pages 103–117, 2023.
[24] Bingyi Zhang et al. BoostGCN: A framework for optimizing GCN inference on FPGA. In *Proc. of FCCM*, pages 29–39, 2021.
[25] Naveen Muralimanohar et al. CACTI 6.0: A tool to understand large caches. In *University of Utah and Hewlett Packard Laboratories, Tech. Rep*, 2009.
[26] George Karypis et al. Metis-unstructured graph partitioning and sparse matrix ordering system, version 2.0. In *University of Minnesota, Tech. Rep*, 1995.