A Review on Important Issues in GCN Accelerator Design

Siyuan Miao

[1]Department of Electrical Engineering, Xi'an Jiaotong University, Xian, Shaanxi Province, 710049, China
**Corresponding author. Email: msy011002@stu.xjtu.edu.cn*

ABSTRACT
Graph convolutional neural networks (GCNs) emerge as an efficient method to process real-world graph data and have been proved powerful in various areas like link prediction and crack detection. While the graphs in GCNs have dynamic and irregular inherent patterns, traditional hardware architectures have poor performance on GCN models and accelerators are needed. This review discusses four bottlenecks for traditional hardware which are also important issues in GCN accelerator designs. The irregular patterns of the input matrix harm the utilization of processing elements (PEs), especially for systolic array-based architectures, which can be alleviated by adopting flexible execution patterns. Then there is a problem of high adjacent matrix sparsity which decreases performance. Usual solutions include using flexible loading patterns and preprocessing adjacent matrix to reduce sparsity. The imbalanced workload in the aggregation stage makes PE utilization drop to as low as 18.3%, increasing processing latency. Therefore, a specially designed hardware architecture that enables the exchange of workloads may be efficient. Last, this review discusses the data reuse problem, which is crucial for saving memory resources. Inner product, outer product and some useful techniques are mentioned.

Keywords: *Graph convolutional neural network, hardware acceleration, accelerator architecture, sparse matrix multiplication*

1. INTRODUCTION

Deep learning networks such as Convolutional Neural Networks (CNNs) and Graph Neural Networks (GNNs) have been proved powerful in a wide range of areas. Inspired by this, Graph Convolutional Networks (GCNs) are proposed particularly to process graph data. GCNs use nodes and edges to represent the structure and properties of graphs, respectively, before constructing a neural network for training and inference. Recently, GCNs have attracted researchers from a large variety of areas to deal with problems including link prediction[1], movement recognition[2], crack detection[3] and traffic prediction[4].

With the development and application of GCNs, designing high-performance and energy-efficient GCN accelerators is becoming an increasingly important issue. Existing hardware architectures like CPUs and GPUs cannot handle the dynamic and irregular inherent patterns of GCNs, resulting in decreased PE utilization and low performance. Furthermore, early hardware designs, including the Systolic Array design, are unable to fully exploit the adjacent matrix's high sparsity. Moreover, the serious workload imbalance problem in the aggregation stage in GCNs increases execution latency a lot. As Moor's Law is slowing down, the size of on-chip memory is not capable of storing all input data in GCNs, especially for some big models. So the issue of data reuse is becoming increasingly critical. Under this circumstance, domain-specific hardware designs of GCN accelerator are needed. Thus, GCN accelerators need to solve the bottlenecks that limit the performance of traditional hardware architectures.

This review first introduces GCN and its execution patterns, as well as the sequential problems for traditional hardware architectures. Then some possible solutions and state-of-the-art techniques against these problems are introduced. After that, this review gives a general perspective of the future development of GCN accelerators.

2. BRIEF INTRODUCTION OF GCN ALGORITHM

A typical GCN algorithm contains three key stages: combination, aggregation and feature update. The combination (also known as feature extraction) stage applies a multilayer perceptron (MLP) neural network, which combines the properties of each vertex and generates a new feature vector. For a given graph, the parameters of the MLP network remain the same for all vertices. Then the aggregation stage aggregates a group of feature vectors from each vertex and its neighbors to one feature vector. This process depends highly on the graph structure which decides the number of neighbors of a vertex. Generally, a GCN algorithm can be presented by Equation 1:

$$X^{k+1} = \sigma(AX^kW^k) \quad (1)$$

In equation 1, $X \times W$ represents the combination stage while $A \times (XW)$ denotes the aggregation stage. In some cases, there is a sample stage before the aggregation stage to choose parts of neighbor vertices of a vertex as new neighbors. And there is a pooling stage after the combination stage to minify the size of the graph. These will not be discussed due to the space limitations.

3. THE ISSUES OF DYNAMIC AND IRREGULAR PATTERNS

3.1. Issues

The aggregation stage does the matrix multiplication $A \times (XW)$. It is mentioned before, that the adjacent matrix A relies highly on the input graph, so there is a big difference between two unrelated graphs and there are no regular node connection patterns. Thus, the execution pattern in the aggregation stage is dynamic and irregular [6].

Performance of existing hardware architectures is limited by these characters. Because memory access is unpredictable, CPUs cannot handle large amounts of dynamic and irregular data access. For GPUs which are suitable for neural networks, they are also unable to deal with the irregularity. This results in a large consumption of time when loading data both for CPUs and GPUs. For the famous design systolic array, which is adopted by Google TPU, the irregular matrix harms the utilization of processing elements (PEs) a lot because of its rigid structure.

3.2. Possible Solutions and Techniques

In the aggregation stage, it is possible to get the graph information before execution, so one method to overcome irregularity is to build a flexible architecture that allows the redistribution of PE workforce according to the input graph information. This idea is widely adopted by state-of-the-art GCN accelerators although they have very different hardware designs.

To be specific, the aggregation engine in HyGCN [6] does not use the vertex-concentrated execution mode: workloads of one vertex are assigned to one single SIMD core (computing unit in HyGCN). In this mode, some SIMD cores may have to wait for those SIMD cores with heavier workloads. Instead, it uses the vertex-disperse mode and distributes the workloads of each vertex to all SIMD cores. In this way, however the input graph changes, all the computing units can participate in the aggregation stage and the performance is improved.

Another typical example is SIGMA [7]. SIGMA is designed for general matrix-matrix multiplications (GEMMs) and it can be used in the calculation of GCNs. The design of SIGMA is based on the systolic array, but the reduction network in SIGMA makes it flexible. Assume that we are facing an irregular adjacent matrix that can only fill the upper half of all the rows of the systolic array. In traditional systolic array design, the partial sums are accumulated downside through the forwarding network, so the data-loading pattern is fixed. If we try to use the disengaged lower half of the PE array and load data into it, errors will take place when accumulating partial sums since calculating results in the lower half are wrongly added to partial sum. In SIGMA, a dedicated designed reduction network replaced the traditional forwarding network. The computation results are accumulated through the reduction network so the unengaged PEs can also be used without accumulation errors. The cost is that additional accumulation cycles are needed. To deal with this, SIGMA adopts the topology of adder-trees in its reduction network and minimizes this cost. According to our simulation, the accumulation cycle only takes 5% of all the processing cycles.

In general, dealing with dynamic and irregular adjacent matrixes necessitates flexibility. If the GCN accelerator's design can transform in response to changes in input graphs, it will be able to handle a variety of GCN missions.

4. THE ISSUES OF HIGH SPARSITY OF ADJACENT MATRIX

4.1 Issues

Another character of aggregation stage in GCNs is that the graph adjacent matrix is usually very sparse. Common datasets like Cora, Citeseer and Pubmed have an adjacent matrix density of lower than 1% [5]. For some datasets like NELL, the density is even under 0.01%. In this case, utilizing this high sparsity is crucial for improving performance.

Some old designs for matrix multiplication cannot use the sparsity and this problem is particularly serious for systolic-array-based designs. Systolic array does not have a good performance on sparse-sparse matrix multiplication or sparse-dense matrix multiplication because of the stationary data loading pattern and rigid forwarding network between PEs, which is not compatible with the dynamic and sparse data pattern. As a result, the utilization rate of processing elements (PEs) is very low. We designed a cycle-accurate simulator to simulate the performance of systolic array on Cora, Citeseer and Pubmed and the result shows that PE utilization is all below 10% on the three datasets. The low PE utilization caused by sparsity is a waste of computing power that needs to be improved.

4.2 Possible Solutions and Techniques

One possible solution is to use a flexible data-loading pattern instead of a stationary one. In the original systolic array, every element of the matrix has to be loaded into the PE array even if the value of the element is zero and no calculation is needed. In contrast, SIGMA [7] adds a dedicated distribution network and makes it possible to load any element of the matrix to any position in the PE array. Also, an element filter is added so only non-zero elements are loaded. In this way, the sparsity of matrix is fully used. However, the cost is that there will be extra loading cycles because of the limited bandwidth and irregular memory access. Our simulation shows that the loading latency takes about 50% of the processing latency, which is not negligible.

5. THE ISSUES OF IMBALANCED WORKLOAD

5.1 Issues

It is mentioned that the graph patterns in GCNs are dynamic and irregular. Specifically, different vertices may have a different number of neighbors due to the irregular connection between vertices. Figure 1 shows the distribution of non-zero elements in the adjacent matrix. For some designs in which one processing element is responsible for the aggregation of one vertex, the workloads of PEs are extremely imbalanced. In these cases, the PEs with light workloads have to wait for PEs with heavy workloads until aggregation of all the vertices is finished. In other words, the processing latency equals to the latency of the slowest PE. As a result, PE utilization is harmed and performance drops.

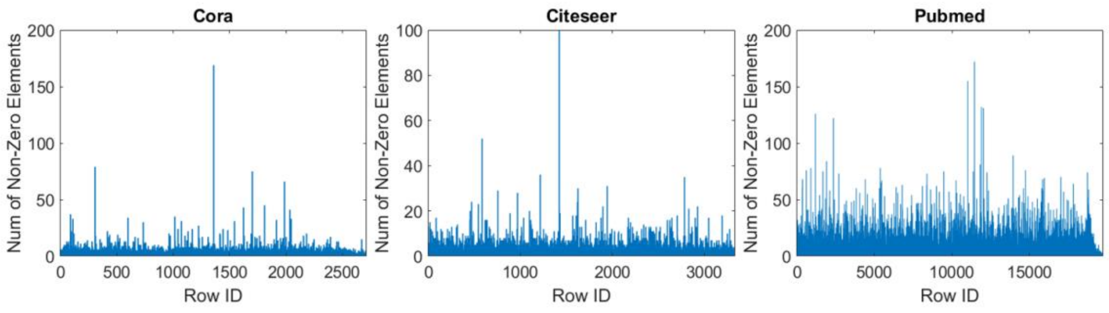

Figure 1 Distribution of none-zero elements in Cora, Citeseer and Pubmed

5.2 Possible Solutions and Techniques

To deal with the imbalanced workload, the natural idea is to balance the PE workload. The best situation is that all PEs have the same workloads, no PE is free during the aggregation process and the PE utilization is 100%.

To reach this situation, some of the latest designs choose to redistribute workloads dynamically during execution. This is based on the fact that when the computation order in the aggregation stage is $A \times (X \times W)$, XW is dense so the workloads of PEs are mainly decided by matrix A which remains constant. This character leads to similar workload distribution in the calculation of the multiplication of A and (part of) XW, whatever XW exactly is. So the workloads of the next calculation can be adjusted according to the workload of the current calculation.

Under this condition, AWB-GCN [5] proposed three hardware-based techniques to handle different kinds of workload clusters: distribution smoothing, remote switching and evil row remapping. Distribution smoothing balances the workloads among neighbors by tracking the number of pending tasks of each PE and move tasks from busier PEs to their less busy neighbors. Remote switching deals with regionally clustered workloads, where all PEs in a region are busy and distribution smoothing does not work. This technique exchanges the workloads between under-loaded PEs and over-loaded PEs. Evil row remapping is employed on PEs with extremely heavy workloads, so their workloads can be distributed to several pre-specified PEs. Note that remote switching and evil row remapping needs workload distribution information of the current round of calculation and they will not work until the next round.

6. THE ISSUE OF LIMITED ON-CHIP MEMORY AND DATA REUSE

6.1 Issues

As Moore's Law is slowing down, the size of the on-chip memory can no longer meet the needs of GCN models whose size is increasing. As a result, not all the data can be put in on-chip memories and some have to be loaded from DRAMs, which takes a large amount of time and energy due to poor locality and irregular memory access patterns. Under this circumstance, data reuse is especially important during calculation. There are different methods to do sparse-dense matrix multiplications and corresponding techniques have been proposed to increase data reuse.

6.2 Inner Product and Its Opitimization

The traditional way to do matrix-matrix multiplication $A \times B = C$ is to use inner product, in which dot product operations are performed between rows of matrix A and columns of matrix B, as shown in figure 2. Thus, each element of matrix C is obtained through a set of multiply-and-accumulate (MAC) operations.

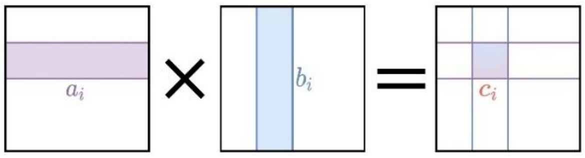

Figure 2 An example of inner product operations

Mathematically:

$$c_{i,j} = \sum_{k=0}^{N-1} a_{i,k} \times b_{k,j} \quad (2)$$

N is the number of columns in matrix A.

Inner product has poor input reuse. To be specific, when doing sparse-dense matrix multiplication in GCNs, the multiplication of A and B is usually decomposed into a series of vector-matrix multiplications, in which entire matrix A multiplies one column of matrix B. Generally, on-chip memory can store one column of matrix B but it may not be big enough for the entire matrix A, especially in some big models. As a result, matrix A has to be loaded more than once (usually N times, N is the number of columns of matrix B), leading to large consumption of energy.

The inner product is usually optimized with graph partition and tiling techniques. Early works [9] propose unaligned block compressed sparse row (UBCSR) format and tiling techniques. Then GraphChi [10] proposes a basic programming model for graph computation on a single machine, followed and optimized by subsequent designs. Among them, NXgraph [11] proposes a scalable partition method, together with the concept of vertex interval and edge shard, which is commonly used [6][12] for graph partitions.

In NXgraph, vertices are partitioned into different intervals and edges are classified according to their source vertex and destination vertex. The size of intervals and shards can be adjusted for different hardware designs. In most cases of GCNs, aggregation is performed interval by interval because this execution order will increase data reuse since vertices in one interval tend to have overlapping neighbors. Also, loading the vertices and edges of adjacent matrix interval by interval can address the problem of poor locality.

6.3 Outer Product and Its Opitimization

Although input reuse of inner product can be improved by graph partition and tiling techniques, the effect is not so satisfying. In order to completely solve the problem of input reuse, OuterSPACE [13] proposes an outer product based design. In outer product multiplication of matrix A and B, each column of A and the corresponding column of B are multiplied to produce N partial matrices (N is the number of columns of matrix A). Then the partial matrices are summed together to form the final result matrix C, as shown in figure 3.

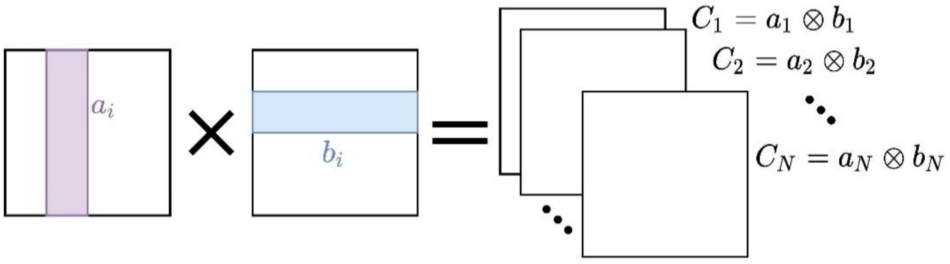

Figure 3 An example of outer product operations

This process can be represented by equation 3:

$$C = \sum_{i=0}^{N-1} C_i = \sum_{i=0}^{N-1} a_i \, b_i \quad (3)$$

N is the number of columns in matrix A.

It is clear to see that each element in matrix A and matrix B is loaded only once in outer product operation. Thus, the input data reuse problem does not exist. However, outer product brings the problem of poor output data reuse. Except for very small models, the on-chip memory cannot hold the considerable amount of partial matrices and they have to be stored in DRAM before the merge phase, resulting in extensive DRAM access. So the outer product-based designs mainly focus on the optimization of merge phase.

One way to deal with this problem is to reduce the total size of partial matrices. SpArch [8] condenses the first input matrix (the original adjacent matrix) to the left, reducing the number of columns, thus reducing the number of partial matrices. Since the adjacent matrix is very sparse, the number of partial matrices is reduced by three orders of magnitude. SpArch also proposes a multiply-merge pipeline and a merge tree so that once partial matrices are computed, they can immediately be sent to merge, instead of waiting in memories. Through these two methods, SpArch achieves good output data reuse.

7. CONCLUSION

In this review, four important issues and corresponding techniques in the design of GCN accelerator are concluded and introduced. First, this paper suggests that a flexible hardware structure is compatible in dealing with irregular and dynamic patterns. Then two methods to solve the high-sparsity problem are mentioned: using flexible data-loading patterns, in the cost of high loading latency or preprocessing input matrix to reduce sparsity, in which sparsity cannot be totally eliminated. After that, this paper discusses the workload imbalance problem and some useful techniques. In the last, the data reuse problem for different execution patterns in matrix-matrix multiplication is introduced and some possible solutions are given.

REFERENCES

[1] X. Wang, J. Wang and Z. Wang, "A Drug-Target Interaction Prediction Based on GCN Learning," 2021 IEEE 9th International Conference on Bioinformatics and Computational Biology (ICBCB), 2021, pp. 42-47, doi: 10.1109/ICBCB52223.2021.9459231.

[2] L. Lo, H. -X. Xie, H. -H. Shuai and W. -H. Cheng, "MER-GCN: Micro-Expression Recognition Based on Relation Modeling with Graph Convolutional Networks," 2020 IEEE Conference on Multimedia Information Processing and Retrieval (MIPR), 2020, pp. 79-84, doi: 10.1109/MIPR49039.2020.00023.

[3] H. Feng et al., "GCN-Based Pavement Crack Detection Using Mobile LiDAR Point Clouds," in IEEE Transactions on Intelligent Transportation Systems, doi: 10.1109/TITS.2021.3099023.

[4] L. Zhao et al., "T-GCN: A Temporal Graph Convolutional Network for Traffic Prediction," in IEEE Transactions on Intelligent Transportation Systems, vol. 21, no. 9, pp. 3848-3858, Sept. 2020, doi: 10.1109/TITS.2019.2935152.

[5] T. Geng et al., "AWB-GCN: A Graph Convolutional Network Accelerator with Runtime Workload Rebalancing," 2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), 2020, pp. 922-936, doi: 10.1109/MICRO50266.2020.00079.

[6] M. Yan et al., "HyGCN: A GCN Accelerator with Hybrid Architecture," 2020 IEEE International Symposium on High Performance Computer Architecture (HPCA), 2020, pp. 15-29, doi: 10.1109/HPCA47549.2020.00012.

[7] E. Qin et al., "SIGMA: A Sparse and Irregular GEMM Accelerator with Flexible Interconnects for DNN Training," 2020 IEEE International Symposium on High Performance Computer Architecture (HPCA), 2020, pp. 58-70, doi: 10.1109/HPCA47549.2020.00015.

[8] Z. Zhang, H. Wang, S. Han and W. J. Dally, "SpArch: Efficient Architecture for Sparse Matrix Multiplication," 2020 IEEE International Symposium on High Performance Computer Architecture (HPCA), 2020, pp. 261-274, doi: 10.1109/HPCA47549.2020.00030.

[9] Vuduc, R W, and Moon, H. Fast sparse matrix-vector multiplication by exploiting variable block structure. United States: N. p., 2005. Web. doi:10.2172/891708.

[10] Y. Jiang, D. Zhang, K. Chen, Q. Zhou, Y. Zhou and J. He, "An improved memory management scheme for large scale graph computing engine GraphChi," 2014 IEEE International Conference on Big Data (Big Data), 2014, pp. 58-63, doi: 10.1109/BigData.2014.7004357.

[11] Y. Chi, G. Dai, Y. Wang, G. Sun, G. Li and H. Yang, "NXgraph: An efficient graph processing system on a single machine," 2016 IEEE 32nd International Conference on Data Engineering (ICDE), 2016, pp. 409-420, doi: 10.1109/ICDE.2016.7498258.

[12] S. Liang et al., "EnGN: A High-Throughput and Energy-Efficient Accelerator for Large Graph Neural Networks," in IEEE Transactions on Computers, vol. 70, no. 9, pp. 1511-1525, 1 Sept. 2021, doi: 10.1109/TC.2020.3014632.

[13] S. Pal et al., "OuterSPACE: An Outer Product Based Sparse Matrix Multiplication Accelerator," 2018 IEEE International Symposium on High Performance Computer Architecture (HPCA), 2018, pp. 724-736, doi: 10.1109/HPCA.2018.00067.