CPU-PCGCN: Efficient Processing of Convolutional Graph Networks on CPU Architectures

Nicolás Meseguer, José L. Abellán,
Manuel E. Acacio

* *Computer Engineering and Technology Department, University of Murcia, 30100, Murcia (SPAIN)*

ABSTRACT

The success of Convolutional Neural Networks (CNNs) in Deep Learning (DL) has led to the extension of the convolutional operation to non-Euclidean data, such as graph-structured citation networks, resulting in Graph Convolutional Networks (GCNs). Given the high computational cost of GCNs, training is typically performed on GPU architectures in large data centers. PCGCN, a software technique implemented on GPU architectures, accelerates this process by dividing the input graph into subgraphs and computing them based on their sparsity. However, in environments with lower performance or without GPU devices, executing PCGCN becomes challenging. This work introduces CPU-PCGCN, an implementation of PCGCN that accelerates GCN computation using the CPU, achieving up to 3.94 times speed increase compared to the base GCN implementation.

KEYWORDS: Convolutional Graph Networks (GCNs); PCGCN; Software Accelerator for Deep Learning

1 Introduction

Deep Neural Networks (DNNs), instrumental in areas such as image recognition and natural language processing, have encountered difficulties with graph data's unique nature, inspiring the development of Graph Neural Networks (GNNs). These specialized networks, particularly Graph Convolutional Networks (GCNs), leverage the concept of convolutional layers in DNNs to transform and aggregate feature information, capturing local graph structures for tasks like vertex classification. In the context of GCNs, vertices are defined by their neighbors and connections, and an iterative process of message passing captures the complex dependencies amongst them. The initial feature vector of a vertex and its neighbors' information are used to create a state encapsulating the local graph structure. However, the computational cost of aggregating these features can be high due to the sparsity of the adjacency matrix, a factor which also poses challenges for efficient graph processing acceleration.

[1]E-mail: {n.meseguerriborra,jlabellan,meacacio}@um.es

The architecture of GCNs, despite being based on spectral graph theory, introduces simplifications for faster training and higher predictive accuracy. They aim to learn a function of signals/features on a graph ***G* = (*V*,*E*)** by taking as input a feature vector $\boldsymbol{x}_{\mathrm{i}}$ for each vertex $\boldsymbol{i}$ and an adjacency matrix $\boldsymbol{A}$, representing the graph structure. After processing, a node-level output $\boldsymbol{Z}$ is produced, where each layer $\boldsymbol{l}$ is a nonlinear function $H^{(l+1)} = f(H^l, A)$ with $f(H^l, A) = \sigma(AH^lW^l)$. Here, W^l is the weight matrix for the ***l*-th** layer of the neural network, and $\sigma(-)$ is a nonlinear activation function like ***ReLu***. GCNs address two main limitations: (1) the inclusion of the vertex's own feature vector in the multiplication with $\boldsymbol{A}$, resolved by adding the identity matrix to $\boldsymbol{A}$, and (2) the normalization of $\boldsymbol{A}$ to maintain the scale of the feature vectors.

Partition-Centric Processing for Accelerating Graph Convolutional Network (PCGCN) [TMYD20] is a software accelerator designed to enhance the computational efficiency of training Graph Convolutional Networks (GCNs) on systems equipped with GPUs. PCGCN adopts a partition-centric method that divides the input graph into subgraphs, adjusting to their sparsity levels and enabling more effective handling during the aggregation stage. By exploiting the graph's locality feature and using different processing modes depending on the sparsity of each subgraph, PCGCN significantly accelerates the propagation stage. For subgraphs with high sparsity levels, a sparse processing mode is chosen, using efficient compressed representations such as COO or CSR. In contrast, for denser subgraphs, a dense processing mode is selected. The operations in the propagation phase of GCNs are modified to depend on each subgraph rather than the entire graph. Feature vectors belonging to the subgraph are collected for each GCN layer in the aggregation phase and processed in a propagation method for each partition. The results are then concatenated to produce the layer output in the combination phase. This approach significantly reduces the computational range and facilitates faster data loading from the cache, leading to overall performance and memory efficiency improvements compared to traditional GCN models.

This work presents CPU-PCGCN[2], a PCGCN implementation designed for low performance systems, which achieves a speed increase of up to 3.94 times compared to the base implementation of PyGCN. CPU-PCGCN is built from a GCN model written in PyTorch, PyGCN [KW16], and uses several contemporary tools like METIS and synthetic graph generators, like Graphlaxy[3] or PaRMAT [KGB15], for a broader study of the tool. Additionally, CPU-PCGCN employs various techniques (task-level parallelism, matrix properties, or even graph representation properties) to accelerate the computation and processing of the model.

2 CPU-PCGCN

This work introduces CPU-PCGCN, an extension of PCGCN aimed at systems consisting only of CPUs, programmed using the popular PyTorch framework. CPU-PCGCN outperforms the baseline (PyGCN) by a factor of up to 3.94 times in the best cases. CPU-PCGCN modifies the graph propagation stage of GCN from a full graph scheme to a subgraph-centric one, which enhances the locality of graph processing. It also introduces a dual computation mode based on the sparsity of a subgraph to further accelerate graph propagation. Our proposal follows a series of well-established steps: (A) processing the datasets; (B) partitioning the network; (C) determining the edge blocks and their sparsity; and (D) running

[2]`https://github.com/NicolasMeseguer/pcgcn`.
[3]`https://github.com/BNN-UPC/graphlaxy`

the model.

A) CPU-PCGCN supports both real-world graph datasets, and synthetic ones, using preprocessed formats for convenience. To address the two significant limitations mentioned in Section 1, the identity matrix I is added to the adjacency matrix A to resolve the first limitation. For the second limitation, the adjacency matrix is made symmetric, $A = A + (A^T \times (A^T > A) - A \times (A^T > A))$, and then normalized so that the sum of the rows equals 1. The feature matrix is also normalized. These techniques resolve the aforementioned issues, with both matrices represented in compressed formats, the first in COO and the second in CSR. Labels are stored in a one-hot vector.

B) CPU-PCGCN employs a 2D graph partitioning technique, dividing the complete graph into K subgraphs, creating K disjoint vertex blocks and $K \times K$ edge blocks where $E_{i,j}$ represents the edges between two subgraphs i and j. While various partitioning algorithms exist, random partitioning methods can harm subgraph locality by assigning vertices randomly. Therefore, this work uses the METIS [KK98] partitioning tool, which requires the input graph and the number K of subgraphs for division. Previous studies have shown that METIS ensures high-quality, uniform partitions, balancing the number of vertices across all partitions.

C) After obtaining the partitions, the formation of edge blocks can be initiated, taking advantage of the symmetry of the adjacency matrix where the upper triangular of the edge blocks is equal to the transpose of the lower triangular. This approach has been demonstrated to achieve a speedup of up to 9 times compared to previous implementations. The sparsity of each subgraph is represented by an integer ranging from 0 to 100, with 100 denoting a fully sparse matrix (i.e., no edges within the edge block).

D) The equivalence between CPU-PCGCN and GCN models has been addressed; CPU-PCGCN only changes the order of neighbor calculations for a specific vertex compared to the original GCN model. In CPU-PCGCN, the locality of graphs is leveraged to accelerate GCN computation by speeding up the propagation phase. Specifically, a centered partitioning scheme (**1**) in the graph propagation stage, enabling locality in graph processing. Additionally, a dual-mode subgraph computation method (**2**) to further accelerate graph propagation by designing calculation modes based on the density of a subgraph.

1. For each GCN layer, the previous hidden state is transformed by a MLP (combination). Then, for each subgraph, it gathers and accumulates states from itself and neighboring subgraphs to execute the graph propagation phase within the partition (aggregation). The outputs are combined as the output of that layer. Since the range of source and destination vertices is limited to the subgraph, this partition-based processing technique maximizes the utilization of the memory hierarchy.

2. Processes GCN from the subgraph perspective. It introduces two modes: selective and complete. Selective mode is used when an edge block has few edges, performing vertex feature multiplication with the scalar value of the edge and adding the result to the corresponding subgraph's hidden state (sparse function, spmatmul). On the other hand, complete mode is employed when a subgraph has many edges, assuming complete connectivity between subgraphs and inserting zero-value edges if necessary (dense function, matmul). To determine the processing mode for each edge block, a hybrid mode is introduced based on the sparsity of the edge block. The computational complexity of each mode depends on the edge block's sparsity.

3 Results

Time (s)							
Datasets	GCN (D)	GCN (S)	CPU-2	CPU-4	CPU-8	CPU-16	Speedup
cora	88.74	81.53	46.74	48.58	56.96	72.53	**1.74x**
citeseer	82.61	69.42	46.11	49.58	56.90	74.97	**1.50x**
pubmed	1,259.93	795.184	248.56	303.22	347.44	446.47	**3.19x**
PaRMAT-0.01	30.70	28.86	19.78	21.71	25.36	33.12	**1.45x**
PaRMAT-0.06	43.01	37.81	21.67	23.70	26.31	34.06	**1.74x**
PaRMAT-0.12	3.76	2.41	1.9	2.04	2.63	4.54	**1.26x**
PaRMAT-1	1,171.2	1,050.3	266.48	229.98	262.54	387.72	**3.94x**

Table 1: The overall execution time (s) of GCN and CPU-PCGCN, denoted as CPU-partitions, for 100 training epochs.

The performance of CPU-PCGCN compared to the baseline PyGCN model implemented in PyTorch demonstrates significant speed enhancements. On average, CPU-PCGCN achieves a speedup of 2.11, peaking at 3.94. Comparisons are made between the base PyGCN implementation using either sparse (S) or dense (D) computation (without partitioning the graph) and CPU-PCGCN with a specific number of partitions, denoted as CPU-partitions. In all cases, CPU-PCGCN shows improvement in computational time. Furthermore, for denser graphs (only for PaRMAT-1), the benefit of partitioning into 4 subgraphs is apparent. However, increasing the partition count to 16 does not improve computation time or cache locality (as seen in PaRMAT-1, where higher partition count leads to higher subgraph density).

4 Acknowledgments

This work was supported by grants TED2021-130233B-C33 and RYC2021-031966-I both funded by MCIN/AEI/10.13039/501100011033 and by the European Union NextGenerationEU/PRTR. Nicolás Meseguer was supported by grant 21803/FPI/22 from Fundación Séneca.

References

[KGB15] Farzad Khorasani, Rajiv Gupta, and Laxmi N. Bhuyan. Scalable simd-efficient graph processing on gpus. In *2015 International Conference on Parallel Architecture and Compilation (PACT)*, pages 39–50, 2015.

[KK98] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM JOURNAL ON SCIENTIFIC COMPUTING*, 20(1):359–392, 1998.

[KW16] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *CoRR*, abs/1609.02907, 2016.

[TMYD20] Chao Tian, Lingxiao Ma, Zhi Yang, and Yafei Dai. Pcgcn: Partition-centric processing for accelerating graph convolutional network. In *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 936–945, 2020.