Random Walk Batching for Cluster GCN on Circuit Congestion Prediction

Spencer Wueste swueste@ucsd.edu
James Bentley jbentley@ucsd.edu
Tommy Shen twshen@ucsd.edu
Srujan Gutta sgutta@ucsd.edu

Lindsey Kostas lkostas@qti.qualcomm.com

Abstract

This research addresses the challenge of efficiently training graph neural networks (GNNs) in the context of system-on-chip (SOC) design. The problem involves batching subgraph partitions for downstream machine learning. Our approach is motivated by the need to reduce computational costs when handling large graphs, particularly complex netlists. We propose a solution strategy that employs heuristic partitioning algorithms in conjunction with a random walk strategy for selecting representative batches for model training. We propose a solution which reduces computational overhead without hindering the overall performance of downstream graph ML applications. The goal is to create a solution that optimizes congestion model training while adhering to the constraints of SOC design.

Code: https://github.com/UCSD-Qualcomm-B07-Group-2/Biased-Batch-Learning-Strategy

1 Introduction

Netlist sub-graph partitioning optimizes the efficiency of the VLSI (Very Large Scale Integration) design flow. We propose a model to reduce compute costs, RAM, and speed up training times on congestion prediction models by using Graph Neural Networks (GNNs) to learn an efficient batching strategy on partitioned netlist data. Our goal is to sample representative clusters that can easily be fed in batches to downstream congestion models.

While various algorithms and machine learning techniques have been explored and designed for social network data, they aren't necessarily effective in chip design. The assumption that neighbors in social networks are inherently related or similar does not apply to netlist graphs. Chip instances that are near in a graph or even neighbors could be vastly different in function thus giving rise to the need for new strategies specific to SoC design. Similarly, social networks tend to consist of undirected graphs whereas the input and output connections of circuit components are vital to their functionalities.

We base our model off of Cluster-GCN but implement an additional random walk step for batching. Cluster-GCN, which was designed with speed in mind, works on pre-partitioned graphs. The concept of partitioning has been widely used across chip design tasks and is a major focus which addresses the increasing complexity of circuits as explained by Moore's Law. We seek a solution that seamlessly combines these partitioning strategies with batching to select representative clusters to produce speedups in training times of predictive models on VLSI graphs.

The strategy we propose is a novel method to batch pre-partitioned graph data, taking advantage of the benefits of partitioning while biasing our batches by neighborhood connectivity.

2 Data

The dataset is a proprietary circuit dataset from a collaboration between the University of California San Diego and North Carolina State University, with support from Qualcomm. There are 13 different designs created from the same data generating process as a larger research-intended circuit known as superblue. We can represent a circuit as a logical representation known as a netlist. We will represent each netlist as a graph, with instances as nodes and nets between instances as edges.

At the highest level, each graph represents a circuit. Below this are global routing cells, which are spatial designations that divide up several regions on a chip. On each global routing cell there may be instances, which we represent as nodes. Each instance contains multiple terminals, which serve as input output sources for nets leading to the other terminals of instances.

3 The Congestion Problem

In a circuit, the two primary considerations for congestion are instances and nets. Nets are able to wire over each other like a bridge. Congestion occurs when there are too many nets in one area. Because our solution is designed for the pre-routing design stage, our data does not have the spatial location of the wires. We therefore use routing demand at the node level as a metric for congestion. Routing demand is the number of nets that pass through an instance, which is a value computed in the post-routing stage. Accurately predicting expected congestion early in the VLSI design process before wire placement will save circuit designers weeks to months of time, identifying poor designs before too much effort has been invested. Due to the wiring placement algorithms being out-of-house and a black box, the process of placement can take from weeks to months at a time. With the time saved, designers are able to reiterate their placement and routing earlier.

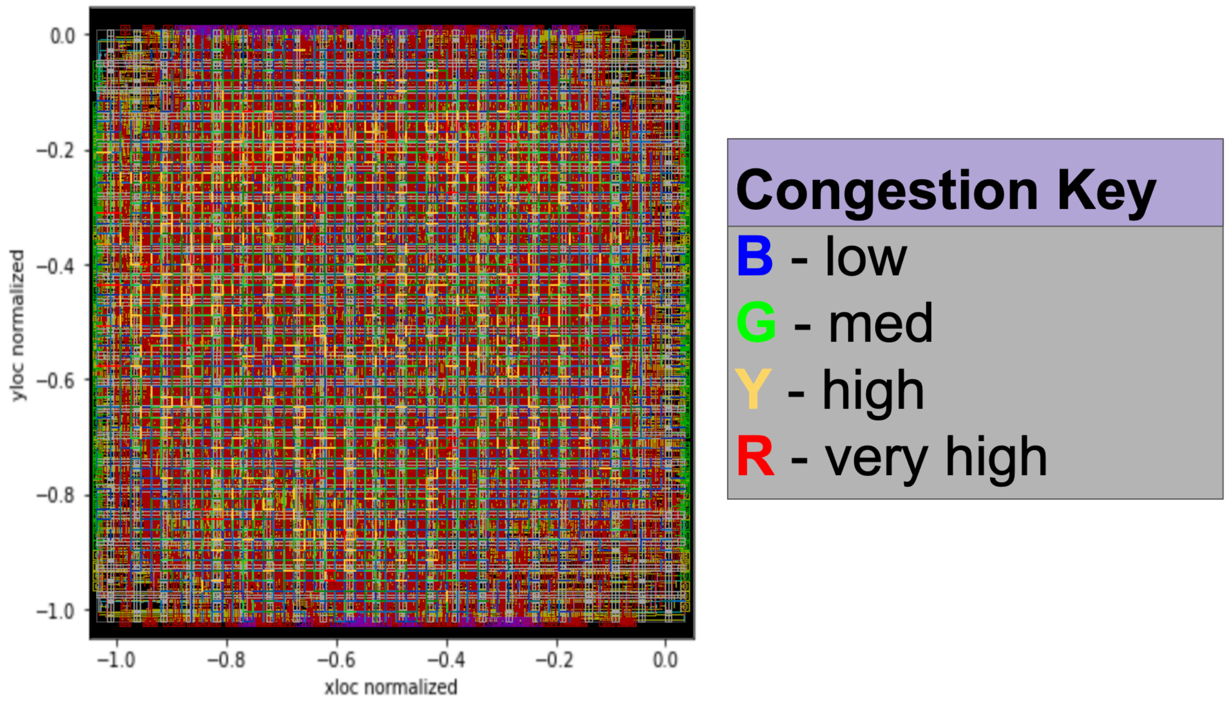

Figure 1: Post-Routing Congestion Visualization of xbar1

4 Partitioning Algorithms

4.1 Kernighan-Lin Algorithm

The Kernighan-Lin (KL) algorithm proposed by Kernighan and Lin (1970) bisects a network into two partitions by iteratively swapping pairs of nodes between the two sets with the goal of minimizing the edge cut between the two sets. Motivated to overcome the the lack of size constraints in the Ford and Fulkerson max flow-min cut algorithm, the KL algorithm aims to produce more balanced partition sizes which are more suitable for applications in paged memory organization and component placement on electronic circuits. The algorithm is explained below.

Let S be a set of $2n$ vertices with an associated cost matrix C. Assume that C is symmetric and $c_i i = 0 \forall i$. The objective is to partition S into sets such that the external cost is minimized. External cost is given by,

$$T = \sum_{A \times B} c_{ab}$$

First, 2 arbitrary partitions, A and B are defined. Initial external cost is reduced by interchanging subsets of nodes between A and B. When no further improvement is possible, the resulting partitions A' and B' is locally minimum. This procedure can be repeated recursively to create more partitions.

This partitioning method was originally designed specifically for assigning components to circuit boards in a manner that minimized connections between boards but can also be applied to arbitrary graphs.

4.2 *k*-Clique Communities

A clique is defined as a complete subgraph. This means that all vertices in the subgraph connect to all other vertices within that subgraph. In NetworkX, the k-cliques community implementation identifies the unions of all cliques of size k that are reachable through adjacent cliques. The parameter k is the minimum size required for a subset of nodes to be considered a clique.

4.3 METIS

METIS is a set of serial partitioning algorithms performing multilevel recursive bisection, k-way, and multi-constraint partitioning. METIS has been tested in various domains including VLSI. METIS was designed in consideration of speed and memory, thus it is faster and less computationally intensive than many common partitioning algorithms.

5 Cluster GCN

GCN models by themselves are not scalable to large graphs. In full-batch gradient descent, the memory requirements of training are given by O(NFL) where N is the number of nodes, F is the embedding dimension, and L is the number of layers (Chiang et al. 2019). This doesn't include the memory requirements of storing a graph's MxN incidence matrix where M represents the number of nets. The algorithm has a high time per epoch and in general, is slow to converge.

GCN models trained with mini-batch SGD however introduce the neighborhood expansion problem - the loss on a single node at layer L requires the node's neighbor node's embeddings at layer L - 1 which again requires recursive downstream layers. This leads to a time complexity exponential to the GCN depth. This strategy utilizes less memory and converges faster but still has high computational overhead during each epoch.

By splitting a graph into disjoint segments via partitioning, we are able to prevent huge computational overheads of deep message passing. Partitioning also serves as a solution to loading in larger graphs which further assists with scalablity.

The efficiency of a mini-batch algorithm can be characterized by the idea of embedding utilization which is proportional to the number of links between nodes in one batch (within-batch links). ClusterGCN is fed in partitioned graph data that is batched into subgraphs for the convolution model. The original implementation of ClusterGCN was tested using METIS partitioning to optimize the mini-batch embedding utilization.

6 Batching

6.1 Motivation

One problem we found with random sampling was that some clusters that were sampled into a batch had little to no edges between each other in the first place between them. This lack of between-cluster edges raised a concern in our Graph Convolutional Network's (GCN) performance. This is because GCN's rely heavily on message passing, a mechanism that transports information from one node to another. This is crucial for the learning of a GCN. In random sampling, we can batch two clusters together, but if these clusters have no edges between them, then no message passing between the two occur.

6.2 Random Walk

A random walk is a path based on a succession of uniformly random steps. In terms of our batching method, we start at a random cluster (super node), visiting neighboring nodes until the walk is finished or a dead end is hit. Our implementation only visits traverse edges that lead to new nodes. The nodes within the clusters represented by selected super nodes and all edges within and between clusters are reconstructed as a single batch. Figure 2 depicts the full batching procedure.

6.3 Weighted Random Walk

A weighted random walk is similar as above, but choosing edges to traverse based off an arbitrarily fixed distribution. In our batching strategy, that would be our edge weights of between-cluster edges. Similar to a uniform random walk, we will visit random nodes

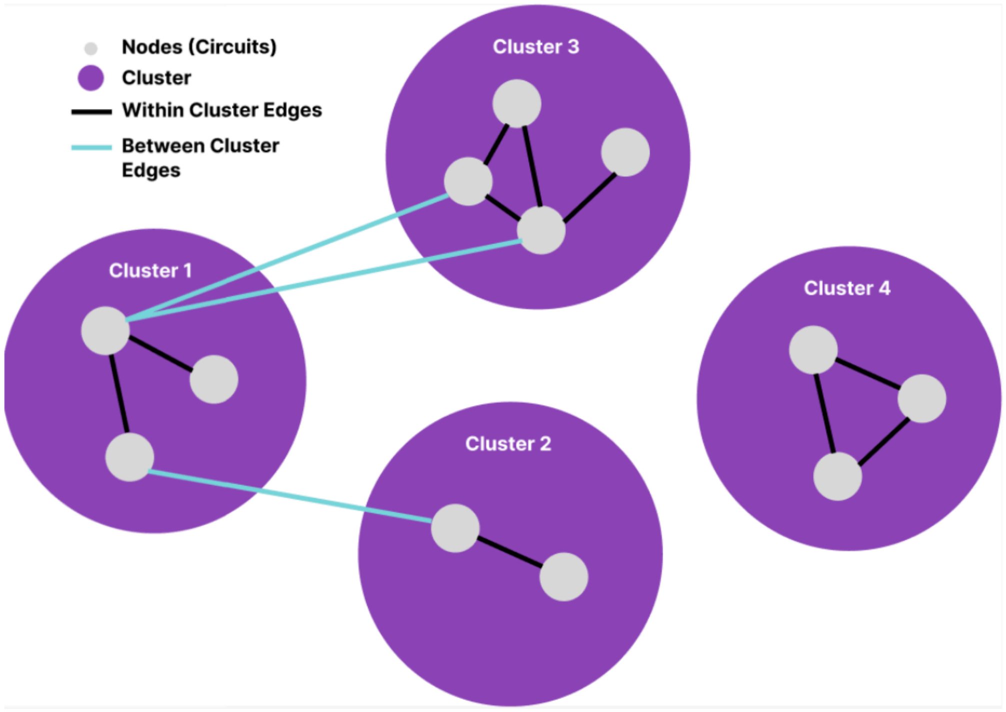

Figure 2: Random Walk Visualization

until the walk is finished or a dead end is hit. This is similar to Markov Chains, with the difference being that our weighted random walk strategy is not memory-less, keeping track of previously visited nodes.

6.4 Modified Weighted Random Walk

We also introduced a second method of weighting our random walks, using neighbor importance. Given a source node U on the super graph random walk, the probability of visiting a target node V was given by the total edge weight of V divided by the sum of all edge weights for all neighbor of U.

7 Congestion Prediction Model

The congestion model is based off a GCN (Graph Convolutional Network), using the features, subgraph centrality (measures the centrality of a node within its subgraphs), local clustering coeff (quantifies how close its neighbors are to forming a complete graph), betweenness (quantifies how close its neighbors are to forming a complete graph), pagerank(represents the overall clustering coefficient for a node, indicating the degree to which nodes in the graph tend to cluster together), eigenvector centrality (scores a node based on the centrality of its neighbors, assigning higher scores to nodes connected to many highscore nodes), degree (number of edges connected to a node), and capacity (the maximum routing resources available for an instance) in order to predict congestion. We perform leave-one-out cross-validation by ..., iterating over all the datasets within each epoch. Beforehand, we were iterating over each dataset for a certain number of epochs before moving on the next dataset, leading to overfitting on the first few datasets. Our leave-one-out strategy ensures the models gradients are more accurate, leading to a more generalize model.

8 Results and Discussion

Our final model with the modified weighted random walk batching strategy, outperforms the random walk strategy's test MSE by 1.84. The weighted random walk batching strategy also slightly outperforms random batching. Though this is slight, we conclude that our solution has the potential to perform better when working with larger datasets such as superblue.

Table 1: Downstream Model Performance Table

Batching Technique Performance			
Batching Technique	Train MSE	Validation MSE	Test MSE
Mean Predictor	N/A	58.73	74.97
Modified Weighted Random Walk	**43.09**	53.25	58.77
Weighted Random Walk	49.10	58.36	58.68
Random Walk	48.36	54.44	60.61
Random Batching (Cluster GCN)	44.59	53.96	59.08
No Batching GCN	43.81	54.53	**48.81**
Random Forest	54.13	**48.00**	59.44

9 Next Steps

As we continue to refine our approach to graph neural network training for SOC design, several key enhancements and expansions are planned to bolster the model's performance, utility, and adaptability. The immediate future work will concentrate on the following areas to complete our model:

1. Testing on larger datasets to assess the scalability of our batching solution.
2. Using current state-of-the-art congestion prediction models for increasing external validity.
3. Experiment using distributed computing cloud services to test the training speedup of our solution when scaling to multiple GPUs.
4. Obtaining edge direction information to feature engineering more complex topological features (pin density and other directional-dependent features) that the model might not find during the training process.

9.1 Model Optimization and Efficiency

We aim to further optimize the computational efficiency and memory usage of our GNN model. This includes investigating advanced optimization techniques, such as quantization and pruning, to reduce model size and accelerate inference without sacrificing accuracy. Additionally, we plan to explore adaptive computation methods to dynamically adjust the model's complexity based on the task's requirements.

References

Chiang, Wei-Lin, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. 2019. "Cluster-GCN: An Efficient Algorithm for Training Deep and Large Graph Convolutional Networks." In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery &amp; Data Mining*. ACM. [Link]

Kernighan, B. W., and S. Lin. 1970. "An efficient heuristic procedure for partitioning graphs." *The Bell System Technical Journal* 49 (2): 291–307. [Link]