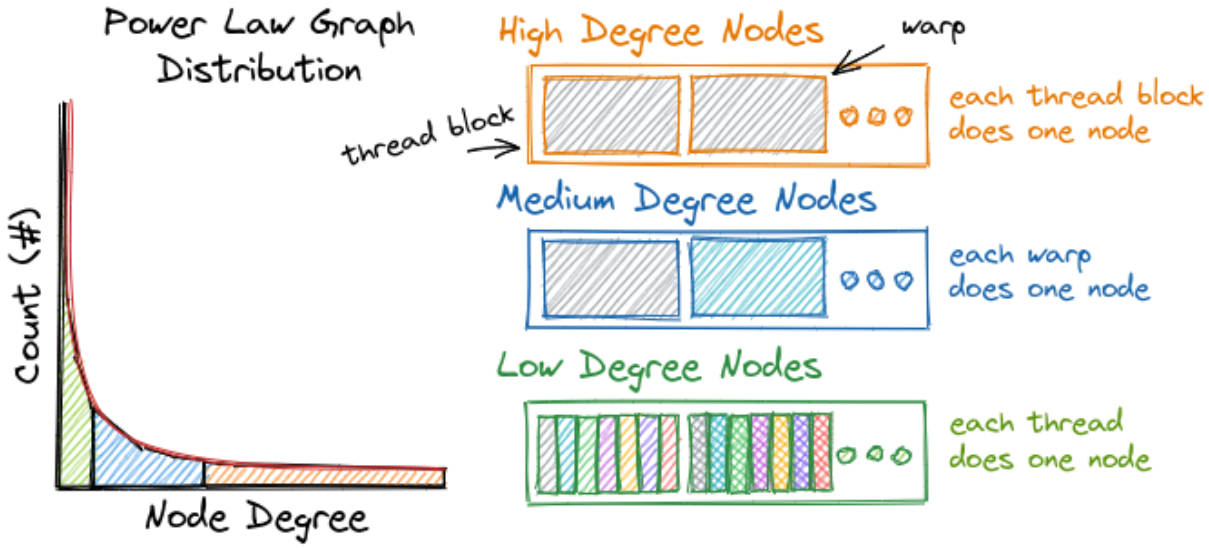
cuAgg: Accelerating GNN Aggregation by Exploiting Graph Structure

Christopher Liu, Rishabh Mani, Karim Saraipour, Scott Zhang

Motivation

Graph Neural Networks are an increasingly popular machine learning model. Graphs allow modeling relationships between different data points. A graph neural network problem can be decomposed into two main patterns: combination (computation) and aggregation (communication). In the combination phase, the aggregated feature vector will be applied to a neural network. This problem has long been studied and high performance libraries (e.g., cuDNN) have been developed. The aggregation phase gathers neighboring feature vectors. Graph processing algorithms are far more difficult to implement on GPUs and arguably have not been able to exploit the architecture to its fullest potential. We look to improve aggregation performance while relying on existing libraries to execute the combination phase. Additionally, due to the limited size of GPU memory, there is also the problem of fitting a large graph along with its associated feature vectors into a single GPU. We also look to implement graph embedding to allow a larger dataset.

Project Idea



Social media graphs (e.g., Facebook, Twitter, LiveJournal) can be characterized as following the power law distribution. This means there are many low-degree nodes and few high-degree nodes. Despite graph structures being inherently irregular, we can take advantage of GPU memory coalescing for medium-/high-degree nodes. If an entire warp operates on a single

node, if the node's degree is high enough, memory accesses may be coalesced even though they may not be perfectly adjacent (accessing the CSR row index array would be coalesced as it's contiguous). For high-degree nodes, we schedule entire thread blocks to perform aggregation for each node; for medium-degree nodes, we schedule a warp for each node; for low-degree nodes, we schedule a thread for each node.

A graph may not fit entirely on a single GPU – especially with the size of feature vectors. We will look into graph decomposition techniques to fit a subgraph into the GPU. Additionally, a stretch goal would be to implement an embedding technique that allows graphs to be dynamically updated without affecting the majority of the structure (i.e., recompute an entire graph embedding every time a node or edge is added).

Evaluation

For evaluation, we'll randomly generate the neural network used during the combination phase as we're only concerned with the performance, not accuracy. Graphs used could be a combination of synthetic Kronecker graphs and real-life datasets. For synthetic graphs, we'll look to use smooth Kronecker graphs since they eliminate the combing degree distribution regular Kronecker graphs exhibit – resembling real world graphs. We will primarily evaluate the performance of the aggregation technique we used.

Levels of Difficulty

Difficulty	Feature
Basic	Basic naive GNN (combination & aggregation)
Basic	High degree node aggregation
Basic	Medium degree node aggregation
Basic	Low degree node aggregation
Advanced	Graph decomposition
Stretch	Advanced graph embedding techniques