Parallel GAT Attention Layer

Senyu Tong
Carnegie Mellon University
Pittsburgh, PA 15213
senyut@andrew.cmu.edu

Yile Liu
Carnegie Mellon University
Pittsburgh, PA 15213
yilel@andrew.cmu.edu

1 Change of Project Focus And Scope

In our project proposal, we intended to deal with the multi-head attention mechanism in Transformer. However, based on the feedback on the proposal, we decided to adjust our focus to allow more space for parallelization.
Now, we will be focusing on Graph Attention Network (GAT), which still utilizes self-attention mechanism but focus on graph-structured data. Comparing to our proposal idea, the possibility of parallelizing on different heads, matrix multiplication still exists and now we can also parallel on different nodes in the network. We can also apply similar ideas from Assignment 3 and 4 on graphs. Based on different structure of graph representation, cache localities will also be interesting to explore. Veličković et al. [2017]

We present an implementation of GAT in C, and we will present optimization / parallelization methods specifically for graph-oriented deep-learning architecture with attention mechanism that could possibly be adapted into commonly-used pyTorch in the future.

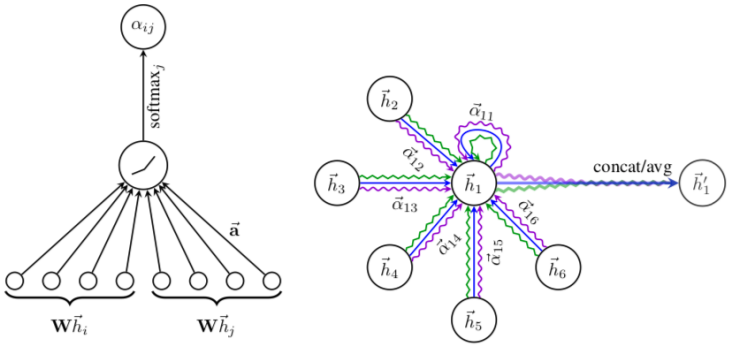

Figure 1: **Left**: The attention mechanism **Right**: An illustration of multi- head attention (with K = 3 heads) by node 1 on its neighborhood. Different arrow styles and colors denote independent attention computations. The aggregated features from each head are concatenated or averaged to obtain $\vec{h}'_t$.

Preprint. Work in progress.

2 Problem Defined - Attention Layer Mechanism

Given a graph with N nodes and m edges, for each node we have node feature $h_i \in R^F$ where F is the dimension of the feature,for each head, we have the two sets of learnable parameters, weights, $W \in R^{F \times F'}$, are used for linear activation to transform the dimension of node features to F'; and self-attentions, $a \in R^{2*F'}$, a vector used in calculating attention coefficients for each edge e_{ij}.

Formally, given $\mathbf{h} = \{h_1, h_2, ..., h_N\}, h_i \in R^F$, and K sets of parameters $\{W^1, ..., W^k\}, W^i \in R^{F \times F'}, \{a^1, ..., a^k\}$, we'll output the transformed embedding $\mathbf{h'} = \{h'_1, h'_2, ..., h'_N\}, h'_i \in R^{KF'}$

$$\alpha_{ij} = \frac{exp(lrelu(a^T[Wh_i||Wh_j))}{\sum_{k \in N_i} exp(lrelu(a^T[Wh_i||Wh_k))} \tag{1}$$

$$h'_i = CONCAT_{k=1,...,K} \sigma(\sum_{j \in N_i} \alpha_{ij}^k W^k h_j) \tag{2}$$

We split the process to 3 steps, the first is to compute linear activation for every nodes, then we get attention coefficients on each edge, and finally get the output embedding. This is reflected as a nested 4-layer for loops and there are many possible optimizations on each layer. Also, since we get each node's neighbors in last two steps, we will test on parallel by partitioning the graph.

3 What we've completed

Project github repo: https://github.com/Senyu-T/Parallel-Graph-Attention-Network-Forward-Phase

We have completed our baseline model, a sequential version of GAT's attention layer in C. We won't use pyTorch since though it has highly optimized vector operations, it has fixed sparse matrix representation form and is not flexible enough to add parallelizatoin on graph-partitioning. We will optimize each of the 3 steps and attention mechanism as a whole instead of each separate linear algebra operation. All our following parallelization will be in C. We have also implemented a python version of GAT as a checker, and the necessary data generaters and loaders.

4 Goals and Deliverable

Our goals have been adjusted accordingly.

- Comparing to our baseline implementation, the sequential version, if we fix the number of heads to be k, our optimization should achieve more than kx speedup. Our ultimate goal will be hitting 2kx speedup.
- We hope we achieve the same speedup as PyTorch implementation.
- We will be testing our implementation on dense/sparse graph, relatively regular and non-regular graphs. The performance on dense and regular graphs should be better.

5 Results

We haven't had preliminary results yet.

6 Presentation and Demonstration

Our presentation will include the following aspects.

- The methods for optimization
- We will use charts to present the result from different approaches. The preliminary chart will look like

Table 1: Results

	Baseline	openMP	MPI	SIMD	...
Graph A					
Graph B					
Graph C					
Graph D					

In the above chart, Graph A/B/C/D will correspond to different types of graphs based on their density and regularity. There will be several charts for different number of heads.

- The best result.
- Other optimization approaches other than loop unrolling, such as optimization on cache localities.

7 Concerns

We have several concerns, but they shouldn't interfere too much with our plan.

- The performance on sparse graph might be the bottleneck of our optimization due to unbalanced workload.
- There are one outer for-loop and three inner for-loops in our implementation. The overhead might be observable.
- We are still trying to figure out an efficient way for parrallelizing both the outer and inner for-loop.
- If we are using MPI, and because the structure of graph is not regular like grid, we are concerned about the balance between communication and computation.

References

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks, 2017.