GPC: Compiler-based Optimization for Sparse Computations in Graph Neural Networks

Yue Jin, Yongchao Liu
Ant Group, China

As the major performance bottleneck, sparse computation acceleration is crucial for highly performant graph neural network (GNN) learning. Recent proposals have mainly focused on optimizing coarse-grained parallelism associated with nodes, edges, and additional feature dimensions; however, important systematic factors such as shared memory tiling, register tiling and load imbalance are vastly overlooked on complex modern DNN accelerators like GPUs. Furthermore, most of these existing optimizations heavily rely on experts' manual engineering efforts which involve much trial and error. To tackle these challenges, we propose GPC, a new compiler framework that extends the popular Halide compiler to enable the effective acceleration for sparse computations for GNNs via compiler-based sparse computation optimizations and cost model based autotuning. Extensive evaluation against highly-optimized state-of-the-art sparse computation kernels and on end-to-end GNN training and inference efficiency has demonstrated that our proposed GPC achieves an up to 3.37× speedup over the state-of-the-art sparse kernels, and a training and inference speedup of 1.44× ∼ 2.33× over three popular GNN frameworks including GCN, GraphSAGE and GAT, with a single Tesla V100 GPU.

Introduction: In GNNs, data is stored in the form of a graph structure, and feature propagation is carried out through this graph structure. Given a graph $G(V, E)$ with node set V and edge set E, let X_u denote the feature of node $u \in V$, and $Y_{e_{u,v}}$ the feature of edge $e_{u,v} \in E$, with u as the source node and v as the destination.

Define the representation of the output node Z_v as $Z_v = \Phi_{u \in N(v)}(X_u \oplus Y_{e_{u,v}})$, where $N(v)$ is the set of inbound neighbors of v, $\oplus$ is a customized operator, and Φ is a customized aggregating operator over $N(v)$. Given that the operation is between a sparse matrix $Y_{e_{u,v}}$ and a dense matrix X_u, the computation of Z_v corresponds to a Sparse Matrix-Matrix Multiplication (SpMM) when $\oplus$ is an add operator and Φ is a sum operator. In contrast, the representation of the output edge is defined as $Z_{e_{u,v}} = lhs \oplus rhs$, where $\oplus$ is a customized operator, and inputs lhs and rhs can be any of X_u, X_v and $Y_{e_{u,v}}$. Given that the output is a sparse matrix with edge features, the computation of $Z_{e_{u,v}}$ is a Sampled Dense Dense Matrix Multiplication (SDDMM) when $\oplus$ is a dot operator.

Motivation: It is well-known that SpMM- and SDDMM-like operations are primary operations in GNNs. However, in practice, graph sampling operations are often observed to dominate the runtimes of GNN training. Typically, two approaches are widely used to remove or hide the overhead of graph sampling. One is to generate subgraph examples beforehand and store them in disk for future use in training/inference. The other is to exploit parallel data prefetching in data loaders. In these cases, the SpMM- and SDDMM-like operations become the bottleneck of improving GNN learning speed, motivating us to accelerate them in this paper.

Implementation: We describe SpMM-like and SDDMM-like algorithms and the corresponding schedules using our extended GPC DSL for CUDA-enabled GPUs. For SpMM-like, we introduce the buffer bound inference technique to enable the 2-D shared memory optimization, the buffer binding index expression to enable the load balancing optimization, and the 1-D stride register tiling to optimize data reuse at the register level. The format to represent the adjacent sparse matrix in SpMM-like is the CSR format. First, the buffer bound inference technique pre-loads the sparse matrix into the shared memory and reuses the data to reduce the global memory access. Second, the buffer binding index expression binds the node of graph, so that the GPC compiler can schedule the GPU blocks to specific nodes with the specific number of neighbors to optimize the balance problem of sparse matrix computations in GNNs. Finally, the 1-D stride register tiling computes multiple outputs in registers, so that we can reuse the data in registers from the sparse matrix.

For SDDMM-like, we use the COO format, and apply the adaptive warp shuffle optimization. Warp shuffle functions use registers, instead of shared memory or global memory, for thread communication within a warp. We assign a specific number of edges to a block, use multiple threads for computing features to intermediate results, and apply adaptive warp shuffles to reduce intermediate results in registers.

We also introduce a cost model based autotuning with an extensive search space to automatically search for optimal results. Our cost model is a simple and trainable DNN. The first layer of our model structure is a `log` function, the second layer is a `batch norm` followed by three `dense` layers, and the last layer is a `normalize` layer. The loss is calculated using mean squared error. The real data is normalized to better reflect the true trend, making $loss = MSE(y_{predicted}, normalize(y))$. A set of metrics representing parallelism, along with GPU block tiling size, shared memory tiling size, register tiling size, and adaptive lane size of warp shuffle, are used to form the search space. In our experiments the tuning process takes 2 ∼ 27 seconds. Moreover, Pearson correlation coefficient shows strong linear relationship between our model predictions and real GPU observations.