6.8 Feature Prefetching

In minibatch training of GNNs, especially with neighbor sampling approaches, we often see that a large amount of node features need to be copied to the device for computing GNNs. To mitigate this bottleneck of data movement, DGL supports *feature prefetching* so that the model computation and data movement can happen in parallel.

Enabling Prefetching with DGL's Builtin Samplers

All the DGL samplers in dgl.dataloading allows users to specify which node and edge data to prefetch via arguments like `prefetch_node_feats`. For example, the following code asks `dgl.dataloading.NeighborSampler` to prefetch the node data named `feat` and save it to the `srcdata` of the first message flow graph. It also asks the sampler to prefetch and save the node data named `label` to the `dstdata` of the last message flow graph:

```
graph = ...                 # the graph to sample from
graph.ndata['feat'] = ...   # node feature
graph.ndata['label'] = ...  # node label
train_nids = ...  # an 1-D integer tensor of training node IDs
# create a sample and specify what data to prefetch
sampler = dgl.dataloading.NeighborSampler(
    [15, 10, 5], prefetch_node_feats=['feat'], prefetch_labels=['label'])
# create a dataloader
dataloader = dgl.dataloading.DataLoader(
    graph, train_nids, sampler,
    batch_size=32,
    ...    # other arguments
)
for mini_batch in dataloader:
    # unpack mini batch
    input_nodes, output_nodes, subgs = mini_batch
    # the following data has been pre-fetched
    feat = subgs[0].srcdata['feat']
    label = subgs[-1].dstdata['label']
    train(subgs, feat, label)
```

Note

Even without specifying the the prefetch arguments, users can still access `subgs[0].srcdata['feat']` and `subgs[-1].dstdata['label']` because DGL internally keeps a reference to the node/edge data of the original graph when a subgraph is created. Accessing subgraph features will incur data fetching from the original graph immediately while prefetching ensures data to be available before getting from data loader.

Enabling Prefetching in Custom Samplers

Users can implement their own rules of prefetching when writing custom samplers. Here is the code of `NeighborSampler` with prefetching:

```
class NeighborSampler(dgl.dataloading.Sampler):
    def __init__(self,
                 fanouts : list[int],
                 prefetch_node_feats: list[str] = None,
                 prefetch_edge_feats: list[str] = None,
                 prefetch_labels: list[str] = None):
        super().__init__()
        self.fanouts = fanouts
        self.prefetch_node_feats = prefetch_node_feats
        self.prefetch_edge_feats = prefetch_edge_feats
        self.prefetch_labels = prefetch_labels

    def sample(self, g, seed_nodes):
        output_nodes = seed_nodes
        subgs = []
        for fanout in reversed(self.fanouts):
            # Sample a fixed number of neighbors of the current seed nodes.
            sg = g.sample_neighbors(seed_nodes, fanout)
            # Convert this subgraph to a message flow graph.
            sg = dgl.to_block(sg, seed_nodes)
            seed_nodes = sg.srcdata[NID]
            subgs.insert(0, sg)
         input_nodes = seed_nodes

         # handle prefetching
         dgl.set_src_lazy_features(subgs[0], self.prefetch_node_feats)
         dgl.set_dst_lazy_features(subgs[-1], self.prefetch_labels)
         for subg in subgs:
             dgl.set_edge_lazy_features(subg, self.prefetch_edge_feats)

         return input_nodes, output_nodes, subgs
```

Using the `set_src_lazy_features()`, `set_dst_lazy_features()` and `set_edge_lazy_features()`, users can tell `DataLoader` which features to prefetch and where to save them (`srcdata`, `dstdata` or `edata`). See 6.4 Implementing Custom Graph Samplers for more explanations on how to write a custom graph sampler.