1.6 Using DGLGraph on a GPU

(中文版)

One can create a `DGLGraph` on a GPU by passing two GPU tensors during construction. Another approach is to use the `to()` API to copy a `DGLGraph` to a GPU, which copies the graph structure as well as the feature data to the given device.

```
>>> import dgl
>>> import torch as th
>>> u, v = th.tensor([0, 1, 2]), th.tensor([2, 3, 4])
>>> g = dgl.graph((u, v))
>>> g.ndata['x'] = th.randn(5, 3)  # original feature is on CPU
>>> g.device
device(type='cpu')
>>> cuda_g = g.to('cuda:0')  # accepts any device objects from backend framework
>>> cuda_g.device
device(type='cuda', index=0)
>>> cuda_g.ndata['x'].device       # feature data is copied to GPU too
device(type='cuda', index=0)

>>> # A graph constructed from GPU tensors is also on GPU
>>> u, v = u.to('cuda:0'), v.to('cuda:0')
>>> g = dgl.graph((u, v))
>>> g.device
device(type='cuda', index=0)
```

Any operations involving a GPU graph are performed on a GPU. Thus, they require all tensor arguments to be placed on GPU already and the results (graph or tensor) will be on GPU too. Furthermore, a GPU graph only accepts feature data on a GPU.

```
>>> cuda_g.in_degrees()
tensor([0, 0, 1, 1, 1], device='cuda:0')
>>> cuda_g.in_edges([2, 3, 4])   # ok for non-tensor type arguments
(tensor([0, 1, 2], device='cuda:0'), tensor([2, 3, 4], device='cuda:0'))
>>> cuda_g.in_edges(th.tensor([2, 3, 4]).to('cuda:0'))  # tensor type must be on GPU
(tensor([0, 1, 2], device='cuda:0'), tensor([2, 3, 4], device='cuda:0'))
>>> cuda_g.ndata['h'] = th.randn(5, 4)  # ERROR! feature must be on GPU too!
DGLError: Cannot assign node feature "h" on device cpu to a graph on device
cuda:0. Call DGLGraph.to() to copy the graph to the same device.
```