Graph Neural Networks on FPGAs with hls4ml

Vladimir Loncar, Jan Schulte, Mia Liu, Phil Harris

Fast Machine Learning for Science workshop
September 25-28 2023
https://indico.cern.ch/event/1283970/

OAC-2117997

https://a3d3.ai/

Introduction

- **Graphs** are often a natural representation of data (**nodes**) and their relation to each other (**edges**)
  - In particle or nuclear physics collider experiments, hits in **charged particle tracking** detectors can be represented as the nodes in a graph
  - Allows for single particle (e.g. particle trajectory) or event level (e.g. rare $\tau \rightarrow 3\mu$ decays) inference using **Graph Neural Networks (GNNs)**
- Use of GNNs in systems with strict latency constraints (e.g trigger systems of sPhenix or CMS) requires **FPGA** implementations
- **Support for GNNs** in tools like **hls4ml** is therefore desirable
- Missing pieces so far:
  - GNNs usually implemented in **pytorch**/**pytorch geometric (PyG)**, only limited pytorch support in hls4ml
  - **Missing support** for several typical operations in GNNs, such as scatter_*
- Presented today is a **prototype** for conversion and HLS code generation of a PyG GNN model in hls4ml

Improved pytorch support

- First step to support PyG model: **Improve general pytorch support** in hls4ml
- Pytorch models are **defined as classes** inheriting from a **“Module” class**. Operations in the model are defined in the **“forward” function**
    - Can be pytorch classes and function, but also **general python operations** such as “getitem” and “getattr”

```python
import torch.nn as nn
import torch.nn.functional as F
class MyModuleBatchNorm(nn.Module):
    def __init__(self):
        super(MyModuleBatchNorm, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=1, out_channels=10,
                               kernel_size=5,
                               stride=1)
        self.conv2 = nn.Conv2d(10, 20, kernel_size=5)
        self.conv2_bn = nn.BatchNorm2d(20)
        self.dense1 = nn.Linear(in_features=320, out_features=50)
        self.dense1_bn = nn.BatchNorm1d(50)
        self.dense2 = nn.Linear(50, 10)

    def forward(self, x):
        x = F.relu(F.max_pool2d(self.conv1(x), 2))
        x = F.relu(F.max_pool2d(self.conv2_bn(self.conv2(x)), 2))
        x = x.view(-1, 320) #reshape
        x = F.relu(self.dense1_bn(self.dense1(x)))
        x = F.relu(self.dense2(x))
        return F.softmax(x)
```

- torch.FX package allows for **symbolic tracing** to capture all model operations
- Rewrote pytorch converter in hls4ml; supports a large set of NN layers
    - Included in master branch, **will enter v0.8**
    - RNN support exists in limited form in a separate branch, to be included
    - Started to work on supporting brevitas models

GNN support in hls4ml

- Parsing of **GNN models in PyG** can use largely the **same converter**
- Extended **operations supported in hls4ml** based on what was needed to implement a GNN developed for track reconstruction in the sPhenix trigger
  - scatter_* operations, such as scatter_add
  - Python operations such as “getitem”
  - “gather” operations and operations such as “ones()” and “zeros()”
- For **more general GNN** support, need to also add support for PyG MessagePassing layers
- **Successfully converted and synthesized** the sPhenix tracking GNN for the first time last week
  - **Large model,** had to be **broken up into pieces**
  - scatter_* implementation **not optimized, large resources usage**

sPHENIX tracking GNN hls4ml synthesis results

Extremely preliminary - DO NOT TRUST NUMBERS

- Network inputs: nodes=80, edges=100
- Input network
  - Can be parallelized to be "nodes" times faster (i.e., 15ns)

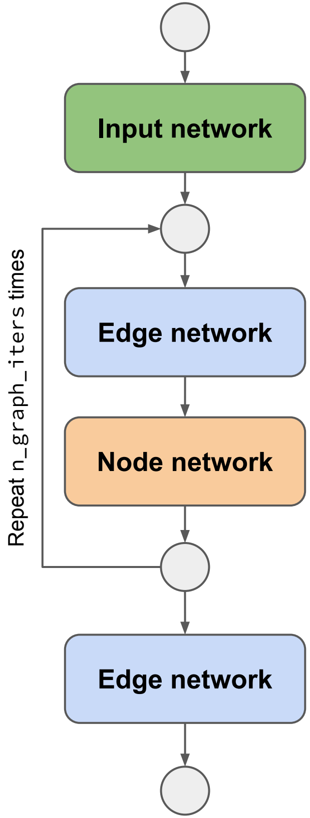
Latency	BRAMs	DSPs	FFs	LUTs
1.2 us	6.5%	0.3%	5%	7.5%

- Edge network

Latency	BRAMs	DSPs	FFs	LUTs
3 us	15%	2%	20%	65%

- Node network (results from HLS synthesis, vivado synthesis OOM'd)
  - Neet to optimize the scatter_add function (expecting ~2us for the net)

Latency	BRAMs	DSPs	FFs	LUTs
12 us	42%	7%	-	-



Outlook

- GNN implementation in hls4ml in prototype stage
  - Currently we know we support one specific model, sort of
  - Significant optimization still necessary
- Need to study how much this can be generalized to other use cases
- Different GNN models using different PyG classes will likely need some adaptation of the converter, possibly also additional HLS code
- Can not all be provided centrally by us, but we are happy to assist users in implementing their needed functionality