GNN-backend in hls4ml

hls4ml: https://github.com/abdelabd/hls4ml/tree/pyg_to_hls_rebase
backend code: https://github.com/abdelabd/hls4ml/blob/pyg_to_hls_rebase/hls4ml/templates/vivado/nnet_utils/nnet_graph.h
Graphs

- $G = (V, E, \text{edge\_index})$;
  - $V[i,j] = \text{jth attribute of ith vertex}$
  - $E[i,j] = \text{jth attribute of ith edge}$
  - edge\_index
    - edge\_index[0,i] = the index of the sending node for the ith edge
    - edge\_index[1,i] = the index of the receiving node for the ith edge
    - or vice versa

- Graph Neural Network = graph-to-graph mapping
  - Input: edge attributes, node attributes, connectivity definition (i.e. edge index)
  - Output: edge attributes, node attributes
Message $\rightarrow$ Aggregate $\rightarrow$ Update

Message:
1. Predict edge/relation attributes
2. Construct “messages”

Aggregate: aggregate all the incoming messages (or edge attributes) for each node

Update: predict node attributes
nnet::EdgeBlock = Message + Aggregate

\[
nnet::\text{EdgeBlock}<\text{data}_T, \text{index}_T, \text{res}_T, \text{typename CONFIG}_T>(\text{node}_\text{attr}, \text{edge}_\text{attr}, \\
\text{edge}_\text{index}, \text{edge}_\text{update}, \text{edge}_\text{update}_\text{aggr}, \text{weights}_0, \text{bias}_0, \ldots,)
\]

- Sends inputs through NN to construct edge_update, aggregates this edge_update on a per-node basis to construct edge_update_aggr
- flow: source_to_target or target_to_source
  - source_to_target
    - edge_index[0,i] = the index of the sending node for the ith edge
    - edge_index[1,i] = the index of the receiving node for the ith edge
  - target_to_source
    - vice versa
- aggregation: sum, mean, or max
  - For each node, aggregate over all the incoming edges to that node
    - e.g. node #3 is the receiving node of edge #7 and edge #8 \rightarrow
      edge_update_aggr #3 is the aggregate of edge_update #7 and edge_update #8
nnet::NodeBlock = Update

- Sends inputs through NN to construct node_update

nnet::aggregate = Aggregate

- No NN involved, just constructs aggregate edge attributes
Sum and mean aggregation

- Let “i” represent the index of any edge, and “r” represent the index of the receiver node for that edge
- `edge_update_aggr` is initialized as an array of zeros
- For each `edge_i`, construct `edge_update_i` and add that to `edge_update_aggr_r`
- If aggregation=add, then we’re done
- If aggregation=mean, divide `edge_update_aggr_r` by the number of incoming edges to `node_r` (using a LUT)
Max aggregation

- Again, let “i” represent the index of any edge, and let “r” represent the index of the receiver node for that edge
- Let \text{MOST\_NEGATIVE\_NUM} represent the most negative number that can be represented in however-many bits your model uses.
- \text{edge\_update\_aggr} is initialized as an array of \text{MOST\_NEGATIVE\_NUM}
- For each edge\_i, construct edge\_update\_i
  - If \text{edge\_update\_i[j]} > \text{edge\_update\_aggr\_r[j]}, update \text{edge\_update\_aggr\_r[j]} = \text{edge\_update\_i[j]}
  - Otherwise, do nothing
- For disconnected nodes, update \text{edge\_update\_aggr\_r} = 0*\text{edge\_update\_aggr\_r}
User-inputs

```python
# model.forward() dictionary
forward_dict = OrderedDict()
forward_dict['R1'] = 'EdgeBlock'
forward_dict['0'] = 'NodeBlock'
forward_dict['R2'] = 'EdgeBlock'

graph_dims = {
    "n_node_max": args.max_nodes,
    "n_edge_max": args.max_edges,
    "node_dim": 3,
    "edge_dim": 4
}
```

```python
def pyg_to_hls(model, forward_dict, graph_dims,
               activate_final = None,
               fixed_precision_bits=16,
               fixed_precision_int_bits=6,
               int_precision_bits=16,
               int_precision_signed=False,
               output_dir = None):
```
Graph padding and truncation

Hardware-implementation requires: n_nodes_max, n_edges_max

1. Truncation: removes true nodes, true edges, or both
   a. Always bad
      i. Fixes: Look for disconnected nodes, remove the least connected nodes first
      1. Compute

2. Padding: adds dummy nodes, dummy edges, or both
   a. sometimes bad
      i. Example: n_nodes >= n_nodes_max, n_edges < n_edges_max
         1. Dummy edges must connect true nodes
   b. sometimes alright
      i. Example: n_nodes<n_nodes_max, n_edges <= n_edges_max
         1. Dummy nodes disconnected, or connected with strictly dummy edges
HLS pragmas

- Inputs and outputs: #pragma HLS ARRAY_RESHAPE complete dim=0
- Intermediate products: #pragma HLS ARRAY_PARTITION complete dim=0
- In front of for-loops: #pragma HLS PIPELINE II=CONFIG_T::reuse_factor
- Inside of for-loops:
  - #pragma HLS UNROLL
  - if (!CONFIG_T::remove_pipelinePragma) {#pragma HLS PIPELINE II=1 rewind}
Interaction Network Benchmarks
### Resource Numbers

- max_nodes = 28, max_edges = 37, n_neurons = 8, fp_bits = 16, reuse = 1

**Vivado Synthesis**

**C Synthesis (Latency = 108*5ns)**

<table>
<thead>
<tr>
<th>Site Type</th>
<th>Used</th>
<th>Fixed</th>
<th>Available</th>
<th>Util%</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLB LUTs*</td>
<td>854583</td>
<td>0</td>
<td>663360</td>
<td>128.83</td>
</tr>
<tr>
<td>LUT as Logic</td>
<td>842479</td>
<td>0</td>
<td>663360</td>
<td>127.60</td>
</tr>
<tr>
<td>LUT as Memory</td>
<td>12104</td>
<td>0</td>
<td>293760</td>
<td>4.12</td>
</tr>
<tr>
<td>LUT as Distributed RAM</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>LUT as Shift Register</td>
<td>12104</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>CLB Registers</td>
<td>340729</td>
<td>0</td>
<td>1326720</td>
<td>25.68</td>
</tr>
<tr>
<td>Register as Flip Flop</td>
<td>340729</td>
<td>0</td>
<td>1326720</td>
<td>25.68</td>
</tr>
<tr>
<td>Register as Latch</td>
<td>0</td>
<td>0</td>
<td>1326720</td>
<td>0.60</td>
</tr>
<tr>
<td>CARRY8</td>
<td>74754</td>
<td>0</td>
<td>82920</td>
<td>90.15</td>
</tr>
<tr>
<td>F7 Muxes</td>
<td>37696</td>
<td>0</td>
<td>331680</td>
<td>11.37</td>
</tr>
<tr>
<td>F8 Muxes</td>
<td>0</td>
<td>0</td>
<td>165840</td>
<td>0.00</td>
</tr>
<tr>
<td>F9 Muxes</td>
<td>0</td>
<td>0</td>
<td>82920</td>
<td>0.00</td>
</tr>
</tbody>
</table>

---

*Utilization Estimates*

<table>
<thead>
<tr>
<th>Name</th>
<th>BRAM18K</th>
<th>DSP48E</th>
<th>FF</th>
<th>LUT</th>
<th>URAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Expression</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FIFO</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Instance</td>
<td>19</td>
<td>10985</td>
<td></td>
<td>3369958</td>
<td>817145</td>
</tr>
<tr>
<td>Memory</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Multiplexer</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Register</td>
<td>71</td>
<td>13954</td>
<td></td>
<td></td>
<td>4741</td>
</tr>
<tr>
<td>Total</td>
<td>90</td>
<td>10985</td>
<td></td>
<td>3369958</td>
<td>822000</td>
</tr>
<tr>
<td>Available SLR</td>
<td>2160</td>
<td>2760</td>
<td>663360</td>
<td>331680</td>
<td></td>
</tr>
<tr>
<td>Utilization SLR (%)</td>
<td>43.98</td>
<td>398</td>
<td>589</td>
<td>247</td>
<td>100</td>
</tr>
<tr>
<td>Available</td>
<td>4320</td>
<td>5520</td>
<td>1326720</td>
<td>663360</td>
<td></td>
</tr>
<tr>
<td>Utilization (%)</td>
<td>2</td>
<td>199</td>
<td>254</td>
<td>12.8</td>
<td>0</td>
</tr>
</tbody>
</table>