Transferable Graph Optimisers for ML Compilers

Zhou et al.
Google
What’s the problem?

1. Device placement
2. Operation Scheduling
3. Operation Fusion
What was previous work, why was it insufficient?

- Device-specific optimisation by compilers (TensorFlow, XLA, Glow, MLIR) – need to have seen device before

- Heuristic methods on each individual problem ie. Auto-tuning etc.

- Other Reinforcement Learning methods:
  - expensive to train,
  - focused on one problem with no knowledge sharing
Learning solutions need resource efficiency, speed, and
To tackle optimisations that affect each other!
A Reinforcement Learning Approach

- GraphSAGE to capture topological information in the computational graph
- Scalable attention network to capture long-ranged dependencies
- Feature modulation to allow specialisation on graph type without increasing parameter numbers

Figure 3: Overview of GO: An end-to-end graph policy network that combines graph embedding and sequential attention. \( N \): Number of Nodes, \( a \): Size of the action space (number of devices, number of priority levels, etc.). Node features are sorted in topological order.
Multiple Dependent Optimisation Tasks

- Recurrent attention layers for each task
- Parameters shared across tasks.

Multi-task policy network that extends GO’s policy network with additional recurrent attention layers for each task and residual connections. GE: Graph Embedding, FC: Fully-Connected Layer, Nxf: fusion action dimension, Fxd: placement action dimension, Nxs: scheduling action dimension.
Evaluation

• Methods
Proximal Policy Optimisation
Large negative reward for bad optimisation
6 different architectures
4 baseline comparisons
Up to 80000 nodes (8-layers)

• Findings

<table>
<thead>
<tr>
<th>Model (#devices)</th>
<th>GO-one (s)</th>
<th>HP (s)</th>
<th>METIS (s)</th>
<th>HDP (s)</th>
<th>Run time speed up</th>
<th>Search speed up</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-layer RNLM (2)</td>
<td>0.173</td>
<td>0.192</td>
<td>0.355</td>
<td>0.191</td>
<td>9.9% / 9.4%</td>
<td>2.5x</td>
</tr>
<tr>
<td>4-layer RNLM (4)</td>
<td>0.310</td>
<td>0.339</td>
<td>0.508</td>
<td>0.331</td>
<td>13.8% / 15.4%</td>
<td>2.3x</td>
</tr>
<tr>
<td>8-layer RNLM (8)</td>
<td>0.330</td>
<td>0.355</td>
<td>OOM</td>
<td>0.371</td>
<td>3.8% / 58.1%</td>
<td>27.8x</td>
</tr>
<tr>
<td>2-layer GNMT (2)</td>
<td>0.301</td>
<td>0.344</td>
<td>0.344</td>
<td>0.222</td>
<td>27.7% / 14.6%</td>
<td>3.6x</td>
</tr>
<tr>
<td>4-layer GNMT (4)</td>
<td>0.350</td>
<td>0.456</td>
<td>OOM</td>
<td>0.432</td>
<td>34% / 25.4%</td>
<td>58.8x</td>
</tr>
<tr>
<td>8-layer GNMT (8)</td>
<td>0.440</td>
<td>0.562</td>
<td>OOM</td>
<td>0.498</td>
<td>21.9% / 36.5%</td>
<td>7.3x</td>
</tr>
<tr>
<td>2-layer Transformer-XL (3)</td>
<td>0.253</td>
<td>0.268</td>
<td>0.57</td>
<td>0.262</td>
<td>20.1% / 17.4%</td>
<td>40x</td>
</tr>
<tr>
<td>4-layer Transformer-XL (4)</td>
<td>0.253</td>
<td>0.27</td>
<td>OOM</td>
<td>0.259</td>
<td>17.4% / 12.6%</td>
<td>26.3x</td>
</tr>
<tr>
<td>8-layer Transformer-XL (8)</td>
<td>0.350</td>
<td>0.46</td>
<td>OOM</td>
<td>0.425</td>
<td>23.9% / 16.7%</td>
<td>16.5x</td>
</tr>
<tr>
<td>Inception (2) &amp;2</td>
<td>0.220</td>
<td>0.212</td>
<td>OOM</td>
<td>0.202</td>
<td>26.6% / 23.9%</td>
<td>13.5x</td>
</tr>
<tr>
<td>Inception (3) &amp;4</td>
<td>0.433</td>
<td>0.331</td>
<td>OOM</td>
<td>0.429</td>
<td>42.1% / 39.6%</td>
<td>31.6x</td>
</tr>
<tr>
<td>AmoebaNet (4) &amp;6</td>
<td>0.194</td>
<td>0.244</td>
<td>0.276</td>
<td>0.213</td>
<td>76.1% / 61.1%</td>
<td>56.8x</td>
</tr>
<tr>
<td>5-stack 10-layer WaveNet (5)</td>
<td>0.317</td>
<td>0.376</td>
<td>OOM</td>
<td>0.354</td>
<td>16.6% / 11.7%</td>
<td>6.6x</td>
</tr>
<tr>
<td>4-stack 36-layer WaveNet (4)</td>
<td>0.659</td>
<td>0.988</td>
<td>OOM</td>
<td>0.721</td>
<td>59.8% / 96.8%</td>
<td>26x</td>
</tr>
</tbody>
</table>

Table 2: Run time comparison between GO-one, human expert, TensorFlow METIS, and hierarchical device placement (HDP) on six graphs (RNLM, GNMT, Transformer-XL, Inception, AmoebaNet, and WaveNet). Search speed up is the policy network training time speed up compared to HDP (reported values are averages of six runs).

<table>
<thead>
<tr>
<th>Model</th>
<th>Speedup</th>
<th>TI-default</th>
<th>SA</th>
<th>GO-one</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMT (2GPU)</td>
<td>2.82</td>
<td>2</td>
<td>1.19 (+0.37)</td>
<td>RNLM (9GPU)</td>
</tr>
<tr>
<td>NMT (4GPU)</td>
<td>4.09</td>
<td>3.54</td>
<td>12.03 (+2.82)</td>
<td>TRF-XL (7GPU)</td>
</tr>
<tr>
<td>NMT (8GPU)</td>
<td>10.47</td>
<td>10.47</td>
<td>12.65 (+2.18)</td>
<td>TRF-XL (4GPU)</td>
</tr>
<tr>
<td>GO-one</td>
<td>1.06</td>
<td>1.06</td>
<td>1.23 (+0.19)</td>
<td>TRF-XL (8GPU)</td>
</tr>
</tbody>
</table>

Table 3: Speedup of each fusion policy normalized to the no-fusion case (reported in %). The number in the parentheses is the improvement of our work over the default fusion.
**Strengths**

- Generalises across different graphs and tasks – move varied set
- Work on entire graph at once instead of just one node at a time – capture long distance dependencies
- Speed-ups
- Scalable – works on >10000 nodes
- Adaptable to different architectures
- 21% improvement over human experts and 18% improvement over the prior state of the art with 15x faster convergence than simulated annealing
Critique

- Their figures make no sense
- Reproducibility
- Loss of explainability
- Mainly putting together existing components (GraphSage, Transformer, etc.) and so limited novelty from an ML perspective
- Poor explanation of why each technique is useful and why decisions were made
Who used it? Where might it be used?

• Author Suggestions:
  • Benchmark evaluation on new hardware
  • Less effort for maintenance when new hardware is released
  • Decrease carbon footprint of machine learning

• Wider usage:
  • Quite new so not much usage yet
  • People are doing similar work:
  • Could use it for finding new strategies for compiler optimisation in general, or to look at relationships between coupled optimisation problems
Discussion

• Would you use a compiler that you can’t explain?

• Is ML compilation in this way substantially different enough from things like autotuning?


