A Hierarchical Model for Device Placement

Azalia Mirhoseini, Anna Goldie, Hieu Pham, Benoit Steiner,
Quoc V. Le and Jeff Dean

Presented by: Deniz Alkan (da626)

19 November 2025

What is device placement?

- Have:
  - (huge) computational graph
  - set of potentially heterogeneous machines that can run the computations
- Need: mapping from computations to devices

Prior Work

- This is fundamentally a graph partitioning problem
- Previous work that uses graph partitioning:
  - Scotch [6]: consortium of partitioning algorithms
  - Scotch-based min-cut
  - Human brain ??? (expert design)
- But wait... we also have to worry about runtime, not just laod balancing. Can we build cost models? This is expensive.
- Let's slap machine learning on it!
- There is previous work on using neural nets and RL for combinatorial optimisation [7, 2]
- Mirhoseini et al.'s previous work also uses RL for device placement optimisation

Device Placement Optimization with RL [5]

- Input: sequence of connected computations
- Output: sequence of operations to run on the available devices

- Method:
  - learn a policy through policy gradients (REINFORCE algorithms [8])
  - sequence-to-sequence model with LSTM and content-based attention
  - pool operations (“co-location”) using heuristics such as sole dependency, and TensorFlow defaults
  - distributed training

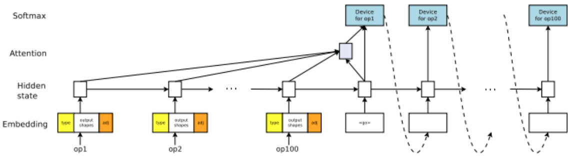

Figure: Model architecture (Fig 2, [5])

- Tricks: moving average baseline, squareroot of runtime as objective, ignore failed iterations – all to reduce variance

The Problem Landscape

What are some problems about prior work?

- need for expert involvement
- co-location based on heuristics
- cannot process large graphs!

The Hierarchical Model

- Two-stage model:
  1. grouping: co-locate operations
  2. placing: assign groups to devices

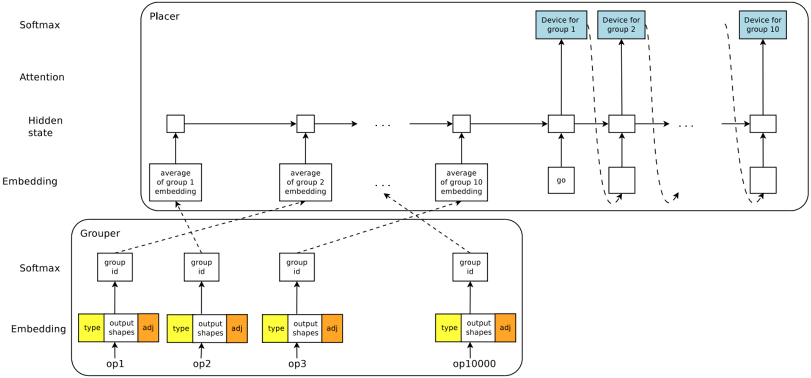

Figure: Hierarchical model architecture (Fig 1, [4])

The Hierarchical Model

Similar structure to prior RL model, intuitive extensions:

- 2 layers of embedding:
  computations $\rightarrow$ embeddings $\rightarrow$ group embeddings
  - fixed-size embeddings: maximum 6 output edges recorded
  - $\#$ of groups $=$ $\#$ of hidden states, fixed as a hyperparameter
- training: the Grouper makes independent predictions and the Placer is conditioned on the Grouper
- "tricks" mentioned previously carry over, alongside distributed training

Evaluation

Tasks	CPU Only	GPU Only	#GPUs	Human Expert	Scotch	MinCut	Hierarchical Planner	Runtime Reduction
Inception-V3	0.61	0.15	2	0.15	0.93	0.82	**0.13**	16.3%
ResNet	-	1.18	2	1.18	6.27	2.92	**1.18**	0%
RNNLM	6.89	1.57	2	1.57	5.62	5.21	**1.57**	0%
NMT (2-layer)	6.46	OOM	2	2.13	3.21	5.34	**0.84**	60.6%
NMT (4-layer)	10.68	OOM	4	3.64	11.18	11.63	**1.69**	53.7%
NMT (8-layer)	11.52	OOM	8	**3.88**	17.85	19.01	4.07	-4.9%

Figure: Runtimes (seconds) with different placements (Table 1, [4])

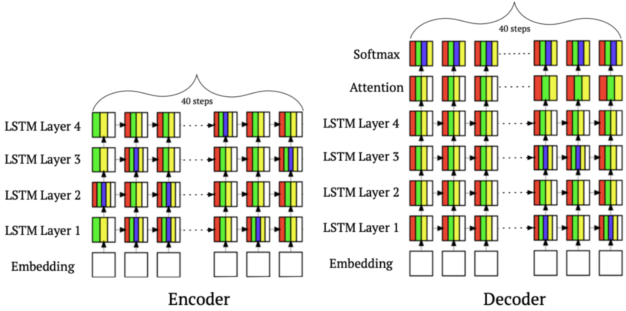

Figure: 4-layer Neural Machine Translation device placement by the Hierarchical model. White: CPU, Colors: different GPUs (Fig 2, [4])

Evaluation

What is the true effect of learning the groups?

Benchmark	Best	Median	Worst	Improvement with Hierarchical Planner
Inception-V3	0.22	0.51	0.65	40.9%
ResNet	1.18	1.18	1.18	0%
RNNLM	1.57	1.57	1.57	0%
NMT (2-layer)	2.25	3.72	4.45	62.7%
NMT (4-layer)	3.20	3.42	6.91	47.2%
NMT (8-layer)	6.35	6.86	7.23	35.9%

Figure: Runtimes (seconds) with randomised groupings (Table 2, [4])

Comparison against no-grouping evaluation is not plotted, but is reported to return successful placements, albeit slowly, for smaller graphs and failures on larger graphs.

Strengths

- good experimental design with evaluations against prior work to demonstrate speedups, and against simplified versions of the hierarchical model (no- and random grouping)
- achieves load balancing as well as runtime improvements
- little-to-no expert help required!
- can process and map very large graphs (demonstrated up to almost 10^5 operations)

Criticisms

- fundamental limitation: any two operations are viewed as similar if they have more than 6 output edges – but neural nets are huge!
- biased evaluation against benchmarks: the model is trained to minimise runtime while Scotch and min-cut promote load balancing and reducing communication overhead. how would the results change if the policy was adapted to reflect this?
- In Mirhoseini et al.'s previous paper, they dismiss the need to design intermediate cost models. However, they empirically choose the square root of the reward. It is not possible to run away from quantifying uncertainty in the model.

Questions and Further Work

- provide device specifications for better predictions or faster learning?
- can we use inductive bias to recognise similar operations (neural networks are symmetric within each layer) and learn to parallelise them?
- run evaluations with TPUs? Would this promote the same level of load balancing?
- New work:
  - *Placeto* [1]: iterative placement improvements + graph embeddings instead of node labels
  - Structure-aware [3]: graph coarsening, node representation learning

Bibliography I

[1] Ravichandra Addanki, Shaileshh Bojja Venkatakrishnan, Shreyan Gupta, Hongzi Mao, and Mohammad Alizadeh. *Placeto: Learning Generalizable Device Placement Algorithms for Distributed Machine Learning*. 2019. arXiv: 1906.08879 [cs.LG]. URL: https://arxiv.org/abs/1906.08879.

[2] Irwan Bello, Hieu Pham, Quoc V. Le, Mohammad Norouzi, and Samy Bengio. *Neural Combinatorial Optimization with Reinforcement Learning*. 2017. arXiv: 1611.09940 [cs.AI]. URL: https://arxiv.org/abs/1611.09940.

Bibliography II

[3] Shukai Duan, Heng Ping, Nikos Kanakaris, Xiongye Xiao, Panagiotis Kyriakis, Nesreen K. Ahmed, Peiyu Zhang, Guixiang Ma, Mihai Capota, Shahin Nazarian, Theodore L. Willke, and Paul Bogdan. *A Structure-Aware Framework for Learning Device Placements on Computation Graphs*. 2025. arXiv: 2405.14185 [cs.LG]. URL: https://arxiv.org/abs/2405.14185.

[4] Azalia Mirhoseini, Anna Goldie, Hieu Pham, Benoit Steiner, Quoc V. Le, and Jeff Dean. "A Hierarchical Model for Device Placement". In: *International Conference on Learning Representations*. 2018. URL: https://api.semanticscholar.org/CorpusID:64663032.

Bibliography III

[5] Azalia Mirhoseini, Hieu Pham, Quoc V. Le, Benoit Steiner, Rasmus Larsen, Yuefeng Zhou, Naveen Kumar, Mohammad Norouzi, Samy Bengio, and Jeff Dean. *Device Placement Optimization with Reinforcement Learning*. 2017. arXiv: 1706.04972 [cs.LG]. URL: https://arxiv.org/abs/1706.04972.

[6] François Pellegrini and Jean Roman. "Scotch: A software package for static mapping by dual recursive bipartitioning of process and architecture graphs". In: *International Conference on High-Performance Computing and Networking*. Springer. 1996, pp. 493–498.

[7] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. *Pointer Networks*. 2017. arXiv: 1506.03134 [stat.ML]. URL: https://arxiv.org/abs/1506.03134.

Bibliography IV

[8] Ronald J. Williams. "Simple statistical gradient-following algorithms for connectionist reinforcement learning". In: *Machine Learning* 8.3 (May 1, 1992), pp. 229–256. ISSN: 1573-0565. DOI: 10.1007/BF00992696. URL: https://doi.org/10.1007/BF00992696.