Distributed Learning - Model Parallelization

Amir H. Payberah
payberah@kth.se
2020-10-26
https://fid3024.github.io
• Train large deep learning models with huge amounts of training data.
• Parallelization and distribution are essential.
Popular Parallelization Methods

[Dean et al., Large Scale Distributed Deep Networks, 2012]
Model Parallelization
The **model** is split across **multiple devices**.

The model is split across multiple devices.

Depends on the architecture of the NN.

NP-Completeness

1. Global Partitioning

2. Local Scheduling

[Mayer, R. et al., The TensorFlow Partitioning and Scheduling Problem, 2017]
Partitioning Approaches

[Mayer, R. et al., The TensorFlow Partitioning and Scheduling Problem, 2017]
Randomly assign vertices to devices proportionally to the capacity of the devices by using a hash function.

[Mayer, R. et al., The TensorFlow Partitioning and Scheduling Problem, 2017]
Assigning the complete **critical path** to the **fastest device**.

**Critical path:** the path with the **longest computation time** from source to sink vertex.

---

[Mayer, R. et al., The TensorFlow Partitioning and Scheduling Problem, 2017]
Model Parallelization - Multi-Objective Heuristics

- Different objectives, e.g., memory, importance, traffic, and execution time

[Mayer, R. et al., The TensorFlow Partitioning and Scheduling Problem, 2017]
ML for Model Parallelization
Device Placement using Reinforcement Learning (1/3)

[Mirhoseini et al., Device Placement Optimization with Reinforcement Learning, 2017]
Device Placement using Reinforcement Learning (2/3)

[Mirhoseini et al., Device Placement Optimization with Reinforcement Learning, 2017]
Device Placement using Reinforcement Learning (3/3)

\[ J(w) = \mathbb{E}_{P \sim \pi(P|G,w)}[R(P)|G] \]

Objective: \( \text{arg min}_w J(w) \)
Device Placement using Reinforcement Learning (3/3)

- \( J(w) = \mathbb{E}_{P \sim \pi(P|G,w)}[R(P)|G] \)
- Objective: \( \arg \min_w J(w) \)
- \( J(w) \): expected runtime

- \( w \): parameters of the RL policy
- \( G \): input neural graph
- \( R \): runtime
- \( P \): output placements
- \( \pi(P|G,w) \): the RL policy (device placement policy)
Device Placement using Reinforcement Learning (3/3)

- $J(w) = \mathbb{E}_{P \sim \pi(P|G,w)}[R(P)|G]$

- Objective: $\arg \min_w J(w)$

- $J(w)$: expected runtime

- $w$: parameters of the RL policy
Device Placement using Reinforcement Learning (3/3)

- \( J(w) = \mathbb{E}_{P \sim \pi(P|G,w)}[R(P)|G] \)
- Objective: \( \text{arg min}_w J(w) \)

- \( J(w) \): expected runtime
- \( w \): parameters of the RL policy
- \( G \): input neural graph
Device Placement using Reinforcement Learning (3/3)

- $J(w) = \mathbb{E}_{P \sim \pi(P|G,w)}[R(P)|G]$
- Objective: $\arg \min_w J(w)$

- $J(w)$: expected runtime
- $w$: parameters of the RL policy
- $G$: input neural graph
- $R$: runtime
Device Placement using Reinforcement Learning (3/3)

- $J(w) = \mathbb{E}_{P \sim \pi(P|G,w)}[R(P)|G]$
- Objective: $\arg \min_w J(w)$

- $J(w)$: expected runtime
- $w$: parameters of the RL policy
- $G$: input neural graph
- $R$: runtime
- $P$: output placements
Device Placement using Reinforcement Learning (3/3)

- \( J(w) = \mathbb{E}_{P \sim \pi(P|G,w)}[R(P)|G] \)
- Objective: \( \arg \min_w J(w) \)

- \( J(w) \): expected runtime
- \( w \): parameters of the RL policy
- \( G \): input neural graph
- \( R \): runtime
- \( P \): output placements
- \( \pi(P|G,w) \): the RL policy (device placement policy)
Device Placement Policy

Grouping → Graph embedding → Device placement
Solution 1

Mirhoseini et al., Device Placement Optimization with Reinforcement Learning, 2017
Mirhoseini et al., A Hierarchical Model for Device Placement, 2018
Device Placement Policy

![Diagram showing the process of Device Placement Policy with intermediate steps marked as 'Not permissible'.]

- Growing
- Graph Embedding
- Device placement

\[ \pi \]
The RL policy is defined as a **attentional seq-to-seq** model.

[Mirhoseini et al., Device Placement Optimization with Reinforcement Learning, 2017]
The RL policy is defined as an attentional seq-to-seq model.

RNN Encoder receives sequence of embedding for each operation.

[Mirhoseini et al., Device Placement Optimization with Reinforcement Learning, 2017]
System Overview

- The RL policy is defined as an attentional seq-to-seq model.
- RNN Encoder receives sequence of embedding for each operation.
- RNN Decoder predicts a device placement for each operation.

[Mirhoseini et al., Device Placement Optimization with Reinforcement Learning, 2017]
Operation Embedding

- The embedding of each operation is the concatenation of its type, its output shape, and its one-hot encoded adjacency information.
Operation Embedding

- The embedding of each operation is the concatenation of its **type**, its **output shape**, and its **one-hot encoded adjacency information**.
- **Type** of the operations, e.g., `MatMul` or `conv2d`. 
Operation Embedding

- The embedding of each operation is the concatenation of its type, its output shape, and its one-hot encoded adjacency information.
- Type of the operations, e.g., MatMul or conv2d.
- The size of each operation's list of output tensors (the output shape).
Operation Embedding

- The embedding of each operation is the *concatenation* of its *type*, its *output shape*, and its *one-hot encoded adjacency information*.
- Type of the operations, e.g., *MatMul* or *conv2d*.
- The size of each operation’s list of output tensors (the *output shape*).
- The one-hot encoding vector that represents the operations that are *direct inputs* and outputs to each operation.
The decoder is an **attentional LSTM** with a fixed number of time steps.

[Mirhoseini et al., Device Placement Optimization with Reinforcement Learning, 2017]
The decoder is an attentional LSTM with a fixed number of time steps.

The number of the steps is equal to the number of operations in a graph $G$. 

[Mirhoseini et al., Device Placement Optimization with Reinforcement Learning, 2017]
RNN Decoder

- The decoder is an **attentional LSTM** with a **fixed number of time steps**.
- The number of the steps is equal to the **number of operations** in a graph $G$.
- At each step, the decoder outputs the **device** for the **operation**.

[Mirhoseini et al., Device Placement Optimization with Reinforcement Learning, 2017]
Training with REINFORCE

\[ J(w) = \mathbb{E}_{P \sim \pi(P|G,w)}[R(P)|G] \]
Training with REINFORCE

- \( J(w) = \mathbb{E}_{P \sim \pi(P|G,w)}[R(P) | G] \)

- \( \nabla_w J(w) = \mathbb{E}_{P \sim \pi(P|G,w)}[R(P) \cdot \nabla_w \log p(P|G,w)] \)

Estimate \( \nabla_w J(w) \) by drawing \( K \) placement samples using \( P \sim \pi(P|G,w) \).

Estimate \( B \): a baseline term to reduce the variance of the policy gradient.
Training with REINFORCE

- \( J(w) = \mathbb{E}_{\mathcal{P} \sim \pi(\mathcal{P}|G,w)}[R(\mathcal{P})|G] \)

- \( \nabla_w J(w) = \mathbb{E}_{\mathcal{P} \sim \pi(\mathcal{P}|G,w)}[R(\mathcal{P})] \cdot \nabla_w \log p(\mathcal{P}|G,w) \)

- Estimate \( \nabla_w J(w) \) by drawing \( K \) placement samples using \( \mathcal{P} \sim \pi(.|G,w) \).
Training with REINFORCE

\[ J(w) = \mathbb{E}_{P \sim \pi(P|G,w)}[R(P)|G] \]

\[ \nabla_w J(w) = \mathbb{E}_{P \sim \pi(P|G,w)}[R(P) \cdot \nabla_w \log p(P|G,w)] \]

- Estimate \( \nabla_w J(w) \) by drawing \( K \) placement samples using \( P \sim \pi(.|G,w) \).

\[ \nabla_w J(w) = \frac{1}{K} \sum_{i=1}^{K} [R(P_i - B) \cdot \nabla_w \log p(P|G,w)] \]
Training with REINFORCE

1. \[ J(w) = \mathbb{E}_{\mathcal{P} \sim \pi(\mathcal{P}|G,w)}[R(\mathcal{P})|G] \]

2. \[ \nabla_w J(w) = \mathbb{E}_{\mathcal{P} \sim \pi(\mathcal{P}|G,w)}[R(\mathcal{P}) \cdot \nabla_w \log p(\mathcal{P}|G,w)] \]

3. Estimate \( \nabla_w J(w) \) by drawing \( K \) placement samples using \( \mathcal{P} \sim \pi(\cdot|G,w) \).

4. \[ \nabla_w J(w) = \frac{1}{K} \sum_{i=1}^{K}[R(\mathcal{P}_i - B) \cdot \nabla_w \log p(\mathcal{P}|G,w)] \]

5. Estimate \( B \): a baseline term to reduce the variance of the policy gradient.
Shortcomings of the Proposed Model

- Seq-to-seq models cannot be unrolled for more than few hundred steps.
Shortcomings of the Proposed Model

- *Seq-to-seq* models cannot be unrolled for more than few hundred steps.

- Most TensorFlow graphs contain tens of thousands of operations.
Shortcomings of the Proposed Model

- **Seq-to-seq** models cannot be unrolled for more than few hundred steps.

- Most TensorFlow graphs contain tens of thousands of operations.

- Manual grouping of operations hampers scalability.
Device Placement Policy

Grouping → Graph Embedding → Device placement

π
An End-to-End Hierarchical Placement Model

- **Grouping operations.**
- **Prediction is for group placement, not for a single operation.**

[Mirhoseini et al., A Hierarchical Model for Device Placement, 2018]
Hierarchical Device Placement Optimization (1/2)

- \( J(w_g, w_d) = \mathbb{E}_{P(d, w_g, w_d)}[R_d] = \sum_{g \sim \pi_g} \sum_{d \sim \pi_d} p(g, w_g)p(d|g, w_g)R_d \)

- Objective: \( \arg \min_w J(w) \)
Hierarchical Device Placement Optimization (1/2)

- \( J(w_g, w_d) = E_{\mathcal{P}(d, w_g, w_d)}[R_d] = \sum_{g \sim \pi_g} \sum_{d \sim \pi_d} p(g, w_g)p(d|g, w_g)R_d \)

- Objective: \( \arg \min_w J(w) \)

- \( G \): input neural graph
Hierarchical Device Placement Optimization (1/2)

- \( J(w_g, w_d) = \mathbb{E}_{P(d, w_g, w_d)}[R_d] = \sum_{g \sim \pi_g} \sum_{d \sim \pi_d} p(g, w_g) p(d | g, w_g) R_d \)

- Objective: \( \text{arg min}_w J(w) \)

- \( G \): input neural graph
- \( R \): runtime
Hierarchical Device Placement Optimization (1/2)

- $J(w_g, w_d) = \mathbb{E}_{P(d, w_g, w_d)}[R_d] = \sum_{g \sim \pi_g} \sum_{d \sim \pi_d} p(g, w_g)p(d|g, w_g)R_d$

- Objective: $\arg\min_w J(w)$

- $G$: input neural graph
- $R$: runtime
- $J(w_g, w_d)$: expected runtime
Hierarchical Device Placement Optimization (1/2)

- \( J(w_g, w_d) = \mathbb{E}_{P(d, w_g, w_d)}[R_d] = \sum_{g \sim \pi_g} \sum_{d \sim \pi_d} p(g, w_g) p(d | g, w_g) R_d \)
- Objective: \( \arg \min_w J(w) \)

- \( G \): input neural graph
- \( R \): runtime
- \( J(w_g, w_d) \): expected runtime
- \( w_g \): parameters of the grouper
Hierarchical Device Placement Optimization (1/2)

- \( J(w_g, w_d) = \mathbb{E}_{P(d, w_g, w_d)}[R_d] = \sum_{g \sim \pi_g} \sum_{d \sim \pi_d} p(g, w_g)p(d|g, w_g)R_d \)

- Objective: \( \arg \min_w J(w) \)

- \( G \): input neural graph
- \( R \): runtime
- \( J(w_g, w_d) \): expected runtime
- \( w_g \): parameters of the grouper
- \( w_d \): parameters of the placer
Hierarchical Device Placement Optimization (2/2)

\[ J(w_g, w_d) = \mathbb{E}_{\mathcal{P}(d, w_g, w_d)}[R_d] = \sum_{g \sim \pi_g} \sum_{d \sim \pi_d} p(g, w_g)p(d|g, w_g)R_d \]
Hierarchical Device Placement Optimization (2/2)

\[ J(w_g, w_d) = \mathbb{E}_{P_d(w_g, w_d)}[R_d] = \sum_{g \sim \pi_g} \sum_{d \sim \pi_d} p(g, w_g)p(d|g, w_g)R_d \]

\[ p(g, w_g) \]: the probability of a sample group assignment \( g \) drawn from the Grouper softmax distribution \( \pi_g \).
Hierarchical Device Placement Optimization (2/2)

\[ J(w_g, w_d) = \mathbb{E}_{P(d, w_g, w_d)}[R_d] = \sum_{g \sim \pi_g} \sum_{d \sim \pi_d} p(g, w_g)p(d|g, w_g)R_d \]

- \( p(g, w_g) \): the probability of a sample group assignment \( g \) drawn from the Grouper softmax distribution \( \pi_g \).

- \( p(d|g, w_g) \): the probability of a sample device placement \( d \) drawn from the Placer softmax distribution \( \pi_d \).
Training with REINFORCE

\[ J(w_g, w_d) = \mathbb{E}_{P(d, w_g, w_d)}[R_d] = \sum_{g \sim \pi_g} \sum_{d \sim \pi_d} p(g, w_g) p(d|g, w_g) R_d \]
Training with REINFORCE

- $J(w_g, w_d) = \mathbb{E}_{P(d, w_g, w_d)}[R_d] = \sum_{g \sim \pi_g} \sum_{d \sim \pi_d} p(g, w_g)p(d|g, w_g)R_d$

- $\nabla_{w_g} J(w_g, w_d) = \sum_{g \sim \pi_g} \nabla_{w_g} p(g, w_g) \sum_{d \sim \pi_d} p(d|g, w_g)R_d$
Training with REINFORCE

\[ J(w_g, w_d) = \mathbb{E}_{P(d, w_g, w_d)}[R_d] = \sum_{g \sim \pi_g} \sum_{d \sim \pi_d} p(g, w_g)p(d|g, w_g)R_d \]

\[ \nabla_{w_g} J(w_g, w_d) = \sum_{g \sim \pi_g} \nabla_{w_g} p(g, w_g) \sum_{d \sim \pi_d} p(d|g, w_g)R_d \]

\[ \nabla_{w_d} J(w_g, w_d) = \sum_{d \sim \pi_d} \sum_{g \sim \pi_g} p(g, w_g) \nabla_{w_d} p(d|g, w_g)R_d \]
A Few Words About Graph Embedding

The slides of this part were derived from Jure Leskovec’s slides - Stanford University
Feature Learning in Graphs

Node $u$ maps to a vector in $\mathbb{R}^d$:

$$ f: u \rightarrow \mathbb{R}^d $$

Feature representation, embedding
Why Learn Embedding?

- The goal is to map each node into a low-dimensional space.
Why Learn Embedding?

- The goal is to map each node into a low-dimensional space.
  - Representation for nodes.
Why Learn Embedding?

▶ The goal is to map each node into a low-dimensional space.
  • Representation for nodes.
  • Similarity between nodes indicates link strength.
Why Learn Embedding?

- The goal is to map each node into a low-dimensional space.
  - Representation for nodes.
  - Similarity between nodes indicates link strength.
  - Encodes network information and generate node representation.
Example

[Perozzi et al., DeepWalk: Online Learning of Social Representations, 2014]
Idea: Convolutional Networks

- Goal is to generalize convolutions beyond simple lattices.
- Leverage node features/attributes (e.g., text, images).
Transform information at the neighbors and combine it:
  - Transform messages $h_i$ from neighbors: $w_i h_i$
  - Add them up: $\sum_i w_i h_i$
Real-World Graphs

- But what if your graphs look like this?
GraphSAGE aggregates neighbouring node embeddings for a given node.
GraphSAGE aggregates neighbouring node embeddings for a given node.

The output of one round of GraphSAGE: new node representation for every node in the graph.
GraphSAGE (2/3)

GraphSAGE (3/3)

\[ h^1_{\mathcal{N}(v)} = \max(f^1_{a}(h^1_u), \forall u \in \mathcal{N}(v)) \]
GraphSAGE (3/3)

- $h^1_{\mathcal{N}(v)} = \max(f^1_a(h^1_u), \forall u \in \mathcal{N}(v))$
- $h^{l+1}_v = f^{l+1}_b(\text{concat}(h^l_v, h^l_{\mathcal{N}(v)}))$
GraphSAGE (3/3)

- $h^l_{N(v)} = \max(f^l_a(h^l_u), \forall u \in N(v))$
- $h^{l+1}_v = f^{l+1}_b(\text{concat}(h^l_v, h^l_{N(v)}))$
- $h_v$: the hidden feature of $v$
GraphSAGE (3/3)

- $h^1_{N(v)} = \max(f_a^1(h^1_u), \forall u \in N(v))$
- $h^{l+1}_v = f_b^{l+1}(\text{concat}(h^l_v, h^l_{N(v)}))$
- $h_v$: the hidden feature of $v$
- $f_a$ and $f_b$: dense layers
GraphSAGE (3/3)

- $h^1_{N(v)} = \max(f^1_a(h^1_u), \forall u \in N(v))$
- $h^1_v^{l+1} = f^{l+1}_b(\text{concat}(h^1_v, h^1_{N(v)}))$
- $h_v$: the hidden feature of $v$
- $f_a$ and $f_b$: dense layers
- $N(v)$: the neighbors of $v$
GraphSAGE (3/3)

- $h^1_N(v) = \max(f^1_a(h^1_u), \forall u \in N(v))$
- $h^{l+1}_v = f^{l+1}_b(\text{concat}(h^l_v, h^l_N(v)))$
- $h_v$: the hidden feature of $v$
- $f_a$ and $f_b$: dense layers
- $N(v)$: the neighbors of $v$
- $h_N(v)$: the aggregated feature from the neighbors of $v$
GraphSAGE Shortcoming

- Nodes with the same neighborhoods have the similar embeddings, regardless of their location in the graph?

[You et al., Position-aware Graph Neural Networks, 2019]
By adding anchor sets - we bypass that problem.
Solution 2

Device Placement Policy

Grouping → Graph embedding → Device placement

π
Placeto System Overview

- Graph embedding
- Placement policy network

- Model the **device placement** as Markov Decision Process (MDP).
- **Initial state** $s_0$, consists of $G$ with an **arbitrary device placement** for each node group.

Model the **device placement** as **Markov Decision Process (MDP)**.

**Initial state** $s_0$, consists of $G$ with an **arbitrary device placement** for each node group.

**Action in step** $t$ outputs a **new placement** for the $t$th node in $G$ based on $s_{t-1}$.

---

MDP Formulation (1/2)

- Model the **device placement** as Markov Decision Process (MDP).
- **Initial state** $s_0$, consists of $G$ with an arbitrary device placement for each node group.
- Action in step $t$ outputs a **new placement** for the $t$th node in $G$ based on $s_{t-1}$.
- Episode ends in $|V|$ steps ($V$: set of nodes in $G$).

![Placement improvement MDP steps](image)

Two approaches for assigning rewards.

- Approach 1: assign 0 reward at each intermediate RL step and the negative run time of the final replacement as final reward.
- Approach 2: assign intermediate rewards $r_t = R(P_{st+1}) - R(P_{st})$. 


Two approaches for assigning rewards.

- Approach 1: assign 0 reward at each intermediate RL step and the negative run time of the final replacement as final reward.
Two approaches for assigning rewards.

- Approach 1: assign 0 reward at each intermediate RL step and the negative run time of the final replacement as final reward.

- Approach 2: assign intermediate rewards $r_t = R(P_{st+1}) - R(P_{st})$
Graph Embedding

- Computing **per-group attributes** (a)

Graph Embedding

- Computing per-group attributes (a)
- Local neighborhood summarization (b)

Graph Embedding

- Computing **per-group attributes** (a)
- Local **neighborhood** summarization (b)
- **Pooling** summaries (c)

Placement Policy Network

- Implements the MDP policy using a **three-layer fully connected neural network**.
- Trains it using the **REINFORCE policy-gradient algorithm**.

Graph Representation Matters in Device Placement (1/2)

[Mitropolitsky et al., Graph Representation Matters in Device Placement, 2020]
Graph Representation Matters in Device Placement (2/2)

[Mitropolitsky et al., Graph Representation Matters in Device Placement, 2020]
Solution 3

Zhou et al., A Single-Shot Generalized Device Placement for Large Dataflow Graphs, 2020
Device Placement Policy

1. Graph embedding
2. Device placement

π
GDP System Overview

- Uses a deep RL approach with graph embeddings and a Transformer.

\[
N: \text{number of nodes, } h: \text{hidden Size, } d: \text{number of devices}
\]

[Zhou et al., GDP: Generalized Device Placement for Dataflow Graphs, 2019]
GDP System Overview

▶ Uses a deep RL approach with graph embeddings and a Transformer.

▶ Generalize to unseen graphs.

\[ \text{N: number of nodes, } h: \text{ hidden Size, } d: \text{ number of devices} \]

[Zhou et al., GDP: Generalized Device Placement for Dataflow Graphs, 2019]
GDP System Overview

- Uses a deep RL approach with graph embeddings and a Transformer.
- Generalize to unseen graphs.
- Generates placement for the whole graph in one step, reducing training time.

\[N: \text{number of nodes, } h: \text{hidden Size, } d: \text{number of devices}\]

[Zhou et al., GDP: Generalized Device Placement for Dataflow Graphs, 2019]
Placement Policy Network (1/2)

- Conventional seq-to-seq models usually target short sequences, which requires grouping beforehand.
Placement Policy Network (1/2)

- Conventional seq-to-seq models usually target short sequences, which requires grouping beforehand.

- LSTM used in previous works is slower and more difficult to train than attention-based models.
Conventional seq-to-seq models usually target short sequences, which requires grouping beforehand.

LSTM used in previous works is slower and more difficult to train than attention-based models.

GDP adopts segment-level recurrence introduced in Transformer-XL to capture long-term dependencies.
Conventional seq-to-seq models usually target short sequences, which requires grouping beforehand.

LSTM used in previous works is slower and more difficult to train than attention-based models.

GDP adopts segment-level recurrence introduced in Transformer-XL to capture long-term dependencies.

The key is to cache (with gradient flows disabled) and reuse the hidden states of previous segments.
Placement Policy Network (2/2)

Figure 1: Illustration of the vanilla model with a segment length 4.
[Z. Dai et al., Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context, 2019]
Prior works focus on a single graph only.
Prior works focus on a single graph only.

In GDP, the RL objective is defined to simultaneously reduce the expected runtime of the placements over a set of $N$ dataflow graphs.
Prior works focus on a single graph only.

In GDP, the RL objective is defined to *simultaneously* reduce the expected runtime of the placements over a set of $N$ dataflow graphs.

$$J(w) = \mathbb{E}_{G \sim G, \mathcal{P} \sim \pi(\mathcal{P}|G,w)}[R(\mathcal{P})|G] = \frac{1}{N} \sum_G \mathbb{E}_{\mathcal{P} \sim \pi(\mathcal{P}|G,w)}[R(\mathcal{P})|G]$$
Summary
Summary

- Model parallelization and device placement
- Hierarchical device placement
- Placeto
- GDP
Reference

- Mayer, R. et al., The TensorFlow Partitioning and Scheduling Problem, 2017
- Mirhoseini et al., Device Placement Optimization with Reinforcement Learning, 2017
- Mirhoseini et al., A Hierarchical Model for Device Placement, 2018
- Zhou et al., GDP: Generalized Device Placement for Dataflow Graphs, 2019
Questions?