Accelerate Model Parallel Deep Learning Training Using Effective Graph Traversal Order in Device Placement

Tianze Wang, Amir H. Payberah, Desta Haileselassie Hagos, and Vladimir Vlassov

KTH Royal Institute of Technology, Stockholm, Sweden
{tianzew,payberah,destah,vladv}@kth.se

Abstract. Modern neural networks require long training to reach decent performance on massive datasets. One common approach to speed up training is model parallelization, where large neural networks are split across multiple devices. However, different device placements of the same neural network lead to different training times. Most of the existing device placement solutions treat the problem as sequential decision-making by traversing neural network graphs and assigning their neurons to different devices. This work studies the impact of neural network graph traversal orders on device placement. In particular, we empirically study how different graph traversal orders of neural networks lead to different device placements, which in turn affects the training time of the neural network. Our experiment results show that the best graph traversal order depends on the type of neural networks and their computation graphs features. In this work, we also provide recommendations on choosing effective graph traversal orders in device placement for various neural network families to improve the training time in model parallelization.

Keywords: Device Placement · Model Parallelization · Deep Learning · Graph Traversal Order.

1 Introduction

Recent years have seen the prevalence of Deep Learning (DL) with larger and deeper models with billions of neurons [2, 29]. Together with the performance boost of DL models comes the increasing computation demand for model training. Most solutions seek to parallelize the training on GPU clusters to meet the requirement of computation power. *Data parallelization* [27] and *model parallelization* [29] of DL models are the most common parallelization strategies. In data parallelization, data are distributed among several servers (a.k.a. devices) in a GPU cluster. In contrast, in model parallelization, the DL model is split into multiple parts and distributed among devices. Assigning different parts of a DL model to different devices is known as *device placement*.

Finding the optimal device placement of DL models in model parallelization is a challenging task. It is mainly due to the large search spaces of potential

parallelization strategies, model architectures, and device characteristics [38]. Despite lots of efforts to improve device placements, it still takes a long time for device placement methods to train [1, 3, 4, 19, 38]. The first effort in automating device placement combines global partitioning and local scheduling by using heuristic strategies to first partition the DL model into smaller parts and then determine the execution schedule of neurons within each part [17].

The state-of-the-art device placement methods use a combination of Graph Neural Network (GNN) and Reinforcement Learning (RL) to find the placement of DL models [1,38]. In these solutions, a DL neural network graph is represented as a Directed Acyclic Graph (DAG), in which each node of the DAG represents a single operation or a group of operations, e.g., convolutions. In a typical setting, a GNN takes a DAG of a DL model and its nodes' features as input and generates node embeddings, which summarize the attributes and neighborhood topology of each node [1,38]. An RL agent then processes the node embeddings and uses a policy to predict device placements for all nodes in the DAG on the given device cluster. To this end, the RL agent needs to *traverse* all the nodes in the DAG and learn to propose placements to reduce the training time of the DL model.

Identifying a good *graph traversal order* can decrease the time to train the RL agent and potentially help the RL agent to find better placements to reduce the DL model training time. In this work, we empirically study the relationship between graph traversal orders and the learning efficiency of the RL agent for finding device placement during the training process. We look into six different graph traversal orders and show how they affect the training process of Placeto [1], a state-of-the-art device placement method on three different families of neural networks. Each family of neural networks contains structurally similar DL models [25]. Our initial results suggest that different traversal orders are better suited for different types of neural networks, and the best graph traversal order to use depends on the attributes of the DL model.

We also explain how our traversal order recommendation can be used in DL models built for Remote Sensing (RS) and Earth Observation (EO) applications. RS and EO are domains where there is a need to provide near real-time services and products for global monitoring of planet earth. EO satellites developed over the years have provided an unprecedented amount of data that need to be processed [6, 40]. Model parallelization methods can contribute to these domains by distributing the computation and memory requirement for training large models on large datasets.

Our contributions are summarized as follows.

1. We empirically study the impact of the graph traversal orders on finding the best device placement for the model parallelization of DL models and, consequently, their training times. In this study, we consider different architectures of DL models, such as Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN). Our study shows that different graph traversal orders triumph at finding the best device placements efficiently for different types of DL models.

2. Based on our empirical evaluation of graph traversal orders in device placement for different model parallelization of DL architectures, we summarize and provide guidelines on identifying the best graph traversal order for a given DL model based on its characteristics. For example, we recommend using Breadth-First Search (BFS) traversal order for model parallelization of RNNs with large average degrees to perform device placement.
3. In the context of RS and EO, we show how our methods on identifying the best graph traversal order can be used on different DL models, e.g., CNN models for satellite image classification and RNN models for sequence classification. Choosing a proper graph traversal method in device placement improves the DL models' training time and enables us to train them on larger datasets within a certain (the same) amount of time. The above two-fold benefit enables real-time online training of model parallelization of DL models with time deadlines.

2 Preliminaries

In this section, we discuss the problem formulation of device placement and present Placeto [1] as a state-of-the-art device placement method and show how it uses GNN and RL for device placement. Moreover, we review the graph traversal order methods we use in this work.

2.1 Device Placement

Let $G(V, E)$ be a DAG that represents the computation graph of a neural network. Each node $v \in V$ describes a single computation operation (e.g., convolution) or a predefined small group of operations (e.g., groups of convolutions nearby) that we are interested in predicting its device placement. Each edge $e \in E$ models the data dependencies between the vertices. Let D denote a given device cluster (e.g., GPU clusters) where $d \in D$ characterizes a single device in D. A placement $p : V \rightarrow D$ assigns each node $v \in V$ to a device $d \in D$. Our goal in device placement is to find a placement p to minimize the training time of G (i.e., the DL model) on the given device cluster D while satisfying the memory constraints of every device in the cluster. When given a fixed number of devices, we can treat the device placement task as a classification problem by considering each device identifier as a label. The classification model takes the DAG of a computation graph G as input and classifies every node or group of nodes of G into devices in D.

2.2 Placeto

Placeto [1] models a device placement task as finding a sequence of iterative placement improvements. In each training round, Placeto takes the current placement of the DAG and the representation of one of its nodes as input and predicts that node's placement. Each training episode lasts until the placements of all the

nodes have been updated once. The Placeto method, in general, consists of two parts, (i) using GNN [26] for making node embeddings of the input DAG, and (ii) using RL for assigning nodes of the DAG to devices. The Placeto model's parameters are shared across episodes, allowing to learn placement policies that generalize well to unseen graphs [1]. Below, we elaborate on these two parts in more detail.

Graph Neural Network. What matters in node embeddings of a computation graph is not only the features of the nodes but also their topological relationship. If two connected nodes are placed on two different devices, there will be data transfer between two devices in both the forward and backward path of model training, which is expensive. Things become even more complicated when there are more complex graph and sub-graph structures. For example, convolution blocks [30] contain parallel computation threads that depend on the same node for input data and send the result to another node for intermediate result concatenation, or temporal dependency [8] during the training of recurrent neural networks can also incur a lot of data communication if the nodes that construct the recurrent unit are located on different devices. Thus, it is crucial to consider the relationships among the nodes while making graph representations.

GNN [26] can generate graph embeddings for each node in a given graph that can generalize to unseen graphs. Placeto [1] uses a graph embedding architecture that computes node attributes (e.g., the execution time of operation, total size of output tensor), summarizes the topology of a local neighborhood through message passing, and uses pooling operations for creating a global summary of the entire graph. Mitropolitsky et al. [22] study the impact of different graph embedding techniques on the training time of DL models. By explicitly modeling the relationships between nodes in a computation graph, better placement can be found by auto device placement methods.

Reinforcement Learning. After generating node embeddings of the input DAG, Placeto uses a RL agent to predict the device placement of the DAG's nodes. The RL agent takes the node embeddings of the DAG as input and generates the probability distribution of the current node over candidate devices as the output. During the training process, the RL agent interacts with the training environment and uses the training time of the input DL model as a reward function to guide the training process. Thus, the RL agent aims to minimize the training time of the DL model across different episodes of training. In each episode of training, Placeto updates the placement of each node one time. Placeto uses REINFORCE policy-gradient method [34] to train the RL agent. Placeto also has a simulator to predict the training time of the DL model, which helps to speed up the training process of RL agent by avoiding taking DL model training time measurement of placements on real hardware.

2.3 Graph Traversal Order

Since the device placement problem is treated as a sequential decision-making task [19], we need to convert the computation graph into a sequence of node

embeddings. Placeto formulated the device placement problem as Markov Decision Process (MDP), where the RL agent selects to update the placement for a node in the computation graph in each state. Thus, we need to form a sequence by traversing the computation graph, which is represented as a DAG. Below, we review some of the graph traversal orders on DAG that one can consider using.

Topological. Topological ordering [9] on the DAG of a computation graph defines a graph traversal order such that for every directed edge $u \to v$ from node u to node v, u must appear before v in the traversal order. Topological ordering can be used to represent dependencies in a DAG where we only visit a node once all its dependencies have been met.

Reversed Topological. A reversed topological ordering of a DAG of a computation graph is simply the reversed order of its topological ordering.

Depth-First Search. Depth-First Search (DFS) is a graph traversal method that starts at source nodes (input nodes of a DAG) and explores the DAG as far as possible by continuously visiting the children nodes of the current node first before visiting the sibling nodes. A DFS ordering is an enumeration of the nodes that is a possible output of applying DFS on the graph. A DFS preorder is a list of nodes that are in the order of when they are first visited by DFS. A DFS postorder is a list of nodes that are in the order of when they are last visited by DFS.

Breadth-First Search. Breadth-First Search (BFS) is a graph traversal method that starts at source nodes (input nodes of a DAG) and explores the DAG by first visiting all the sibling nodes of the current nodes before moving to children nodes. A BFS order of a graph is an enumeration of its nodes that is one possible output of applying BFS on the graph.

Lexicographical. Lexicographical order is an order where the strings are placed in order based on the position of each character in the string and their position in the alphabet. For example, given the names of two nodes in a DAG are $a = a_1a_2 \cdots a_k$ and $b = b_1b_2 \cdots b_k$, the order of the name of the two nodes depends on the alphabetical order of the characters in the first place i that a and b differs. If $a_i < b_i$ then $a < b$, otherwise $a > b$.

3 Graph Traversal Orders in Device Placement

In this section, we discuss challenges in device placement and the impact of graph traversal orders.

3.1 Challenges in Device Placement

Finding a good placement for model parallelization is challenging. Most of the state-of-the-art methods use RL to find placements; however, RL agents still require a long time to train before they can find suitable placements. Mirhoseini

et al. [21] find that it takes 12 to 27 hours for their RL method to find the best placement. Although lots of efforts have been made in reducing the complexity of the problem [19], making the training method more efficient [3, 4], and generalizing them better on unseen computation graph [1, 38, 39], the RL agent still needs a long time to train.

One of the challenges in device placement is defining order for nodes in a DAG. Unlike text and image data, the nodes in DAGs reside in a multi-dimensional space that are linked by edges to represent connectivity [36]. Since a node in a graph might have an arbitrary number of edges, it is challenging for a DL model to encode the structural information of a graph. Recent work in graph representation learning [36] has shown that successfully learning structural information of graphs helps better represent them, which in turn leads to performance improvement of downstream tasks that utilize the graph representations. In Placeto [1], the structural information is (partially) reflected in the sequential order that the device placement method iterates through the nodes of the DAG.

Another challenge in device placement concerns the expressiveness of GNN that are used to generate node embeddings. The GNN that are used by state-of-the-art device placement methods mostly follow the message-passing paradigm, which is known to have inherent limitations. For example, the expressiveness of such GNN is bounded by the Weisfeiler-Lehman isomorphism hierarchy [14]. Also, GNNs are known to suffer from over-squashing [32], where there is a distortion of information propagation between distant nodes. Due to these limitations, the node embeddings created by GNN have limited expressiveness. In such cases, different graph traversal orders in device placement can lead to placements with different DL model training time.

3.2 Impact of Graph Traversal Orders

A graph traversal order determines the order where an RL agent learns the placement of each node in a DAG. We believe that a proper graph traversal order can help the RL agent to learn appropriate placements with a lower DL training time faster. One approach to help the RL agent finds better placements is to prioritize learning the placement of important nodes that have more impact on the DL model training time, e.g., to place the nodes with heavy communications first. On the other hand, misplacement of such nodes can lead to longer DL model training time due to extra data communications between different devices. The placement order in a local neighborhood could also play an important role in finding a suitable device placement. Apart from the straggler problem caused by the unbalanced distribution of computation where all the other devices will have to wait for the slowest device, data communication is also challenging.

For example, many of the modern DL models consist of several computation blocks, where each block has multiple parallel threads sharing the same input and whose output should be concatenated to serve as the input for the next computation block [8, 30, 33]. Suppose these computation threads are placed on different devices; thus, the input of a computation block needs to be replicated

Table 1. Computation Graph Datasets Summary

Features	Dataset		
	`nmt`	`ptb`	`cifar10`
#nodes (avg)	179.44	500.75	303.44
#edges (avg)	476.25	1285.44	444.22
node degree (avg)	2.65	2.56	1.47
diameter (avg)	63.13	316.09	95.63
diameter (min, max)	(41, 69)	(216, 450)	(74, 154)

and sent to these parallel threads to perform the computation independently. All the intermediate results from these parallel threads are later concatenated that will serve as the input of the next computation block. If the RL agent can not anticipate the concatenation of results from parallel computation threads, it might misplace the threads on different devices, which incurs a lot of data transfer for the concatenation node. However, if the RL agent learns the placement of the concatenation node first, it can anticipate placements of predecessor nodes in the computation block to better balance computation and communication.

4 Evaluation

In this section, we present the details of the empirical evaluation setup, results, experiment analysis, and guidelines for choosing an effective graph traversal order for a given DL model.

4.1 Datasets

We conduct our experiments on three different datasets `nmt`, `ptb`, and `cifar10` as in [1, 22]. The `nmt` dataset contains 32 variations of Neural Machine Translation (NMT) [35] with different number of unrolled steps. The computation graphs in `nmt` are a family of encoder-decoder networks with attention structures. The `ptb` and `cifar10` are generated using an RL-based method ENAS [25] that finds the optimal subgraph within a larger graph search space. The `ptb` dataset consists of 32 computation graphs for language modeling tasks, and the `cifar10` dataset consists of 32 computation graphs of CNNs for image classification tasks. The nodes of computation graphs are pre-grouped together in all three datasets to reduce graph sizes in the same way as in [21]. The computation graphs in `nmt`, `ptb`, and `cifar10` have on average 180, 500, and 300 nodes. Table 1 summarizes the three datasets.

Overall, the datasets we are studying for device place are similar to the models in EO. For example, CNN models are used for satellite image classification and detection tasks. RNN models are used to learn from time series of satellite data to monitor an area over different times of the year. Based on the results of our empirical evaluation, we provide guidelines on choosing graph traversal orders for EO tasks in Section 4.4.

4.2 Experiment Setup

We implement all the graph traversal orders in Section 2.3 using NetworkX [5] and refer to them as `topo`, `reversed-topo`, `dfs-preorder`, `dfs-postorder`, `bfs`, and `lexico` hereafter for Topological, Reversed Topological, DFS preorder, DFS postorder, BFS, and Lexicographical, respectively. For the implementation of Placeto, we use the implementation provided in [22], which is based on the original implementation [1]. We use the same simulator in the original implementation to simulate the physical execution environment with different numbers of devices that a neural network can be placed on. We use the same graph traversal order in one experiment, and this order is fixed across different episodes that happened in the experiment.

We conduct experiments on the graphs from each of the three datasets with three, five, and eight devices, in line with [22]. We run independent experiments with the same setting (dataset and number of devices) to account for the stochastic and randomness that might lead to differences in experiment results. We compare different settings for the number of repeated runs on a subset of the whole datasets and found that 10 repeated runs offer a good balance between computation load and the reproducibility of the result.

The experiments are run on a standalone benchmark machine with AMD Ryzen Threadripper 2920X 12-Core Processor and 128 GB of RAM. Since we have $3 \times 32 \times 3 \times 6 \times 10 = 17280$ (3 datasets, 32 graphs in each dataset, 3 different number of devices, 6 graph traversal orders, and 10 repeat for each experiment) experiments to run, we use parallel Docker containers that each have one experiment to speed up the process. We give each graph traversal order the same number of training episodes to run, and it approximately takes a few hours to finish each experiment on the CPU. We empirically found that the metrics we measure in the experiment are not sensitive to the number of parallel Docker containers running simultaneously. We use TensorFlow and NetworkX libraries for the experiment, and we refer the readers to this repository[1] for experiment code and the specific version of the libraries and other software settings.

4.3 Results and Analysis

Through the training process of device placement, the RL agent aims to find device placements with lower training times for the input DL models (i.e., the DAGs). However, the device placement processes might have different learning speeds using different graph traversal orders, meaning that the RL agent can find a placement with lower training times for the input DL model faster if it uses a proper graph traversal order.

We empirically observe that the training process of the RL agent is roughly divided into three phases: (i) episodes 1 to 9, (ii) episodes 10 to 19, and (iii) episodes 20 to 49. In the first phase (episode 1 to 9), the RL agent learns efficiently and finds a better placement across different training episodes. This can

[1] https://github.com/bwhub/Graph_Traversal_Order_in_Device_Placement

Table 2. The number of times graph traversal orders find the placement with the lowest training time on the `nmt` dataset.

`nmt`		**Graph Traversal Order**					
#dev	**episode**	`lexico`	`topo`	`dfs-preorder`	`reversed-topo`	`dfs-postorder`	`bfs`
3 dev	9	0	2	0	**22**	1	7
	19	0	0	1	**23**	1	7
	49	0	0	1	**24**	1	6
5 dev	9	0	0	1	**24**	1	6
	19	0	0	2	**26**	0	4
	49	0	0	0	**27**	0	5
8 dev	9	0	0	0	**23**	3	6
	19	0	0	0	**24**	1	7
	49	0	0	0	**24**	3	5

be explained by the fact that the learning process just started, and finding a good enough placement that is better than a random strategy is not very hard. In the second phase (episode 10 to 19), the learning process slows down, and the RL agent cannot always find drastically better placements than in the first phase. This reflects that the learning process plateaus, and we see diminishing returns. In the third phase (episode 20 to 49), the RL agent overcomes the plateau and finds better placements thanks to the more extended training budget and the knowledge learned through the process. In the rest of the experiments, we compare the best placement training time of different graph traversal orders at these three episodes: 9, 19, and 49.

We report the number of times each graph traversal order finds the placement with the lowest training time for the input DL models in the given dataset. Table 2 shows the result of experiments on the `nmt` dataset. Each row shows the number of times each graph traversal order, compared to other graph traversal orders, finds placements with the lowest training time of the DL model at the given training episode (i.e., 9, 19, or 49) and the number of devices (i.e., 3, 5, or 8). Since there are 32 computation graphs of DL models in each dataset, each row in the table should sum to 32. The comparison between training time is based on 10 repeated experiments to minimize random factors in the training process.

As Table 2 shows, in the `nmt` dataset, the `reversed-topo` order dominates and gives the best result. This can be explained by the fact that the `reversed-topo` order considers how intermediate results are concatenated in the DAG. The RL agent can decide the placement of the concatenation operation first. Then it is easier for the RL agent to collocate the input operations nodes to the concatenation node to minimize expensive data transfer and synchronization between devices during training. In such cases, starting from the nodes in the output layers of the DAG also helps. The `dfs-postorder` order does not work well on the `nmt` dataset (unlike on the `cifar10` dataset that we show) as it has a larger average node degree of 2.65 compared to the average node degree of 1.47 of `cifar10`. This increases the effort for the RL agent to collocate the sibling nodes that are

Table 3. The number of times graph traversal orders find the placement with the lowest training time on the `ptb` dataset.

`ptb`		**Graph Traversal Order**					
#dev	**episode**	`lexico`	`topo`	`dfs-preorder`	`reversed-topo`	`dfs-postorder`	`bfs`
3 dev	9	0	1	6	1	0	**24**
	19	0	1	10	1	2	**18**
	49	0	1	8	5	2	**16**
5 dev	9	0	0	3	0	2	**27**
	19	0	1	3	2	2	**24**
	49	0	1	2	6	5	**18**
8 dev	9	0	0	2	1	1	**28**
	19	0	0	2	5	4	**21**
	49	0	0	2	**13**	5	12

Table 4. The number of times graph traversal orders find the placement with the lowest training time on the `cifar10` dataset.

`cifar10`		**Graph Traversal Order**					
#dev	**episode**	`lexico`	`topo`	`dfs-preorder`	`reversed-topo`	`dfs-postorder`	`bfs`
3 dev	9	1	6	**10**	**10**	4	1
	19	3	3	7	8	**11**	0
	49	5	4	5	**8**	**8**	2
5 dev	9	0	**9**	6	6	**9**	2
	19	0	9	3	**10**	8	2
	49	2	6	6	**9**	6	3
8 dev	9	0	8	3	10	**11**	0
	19	1	4	2	8	**17**	0
	49	0	5	1	11	**15**	0

far away in the placement sequence generated using the `dfs-postorder` order. Better collocation of sibling nodes can also potentially explain why the `bfs` order is the graph traversal order that finds the placement with the lowest training time.

Table 3 shows the result of experiments on the `ptb` dataset. The `bfs` order is the graph traversal order that achieves the best learning efficiency on this dataset. This can be explained by the fact that the DAGs in the `ptb` dataset have more nodes and edges than the `cifar10` and `nmt` datasets. There are potentially more sibling nodes that the RL agent needs to consider when performing the placement. Since sibling nodes in a local neighborhood will be put close together in the traversal sequence generated by `bfs`, it is easier for the RL agent to learn to collocate these nodes together to avoid unnecessary data transfer between devices. In this way, the RL agent does not need to worry too much about long-range dependencies in large DAGs.

Table 4 shows the result of experiments on the `cifar10` dataset. Unlike in the `nmt` and `ptb` dataset, where only one graph traversal order dominates the contest for the optimal graph traversal order, the results are more diverse in

the `cifar10` dataset. For example, the `topo` order achieves the best result in experiments on three devices at episode nine, and the `dfs-preorder` on five devices at episode nine. However, most of the time the `reversed-topo` and `dfs-postorder` orders are the best traversal orders to use since they are the orders that find the placements of DL models with the lowest training time (e.g., experiments on five and eight devices). This can be explained by the fact that there are structures of parallel convolutions in the DAG of the `cifar10` dataset where the intermediate results for parallel convolutions are concatenated for later use. In such cases, it is better to start the learning process from the nodes in the output layer of the model. Once the placement of concatenation nodes located near the output layer is settled, it will be easier for the RL agent to optimize the placement for the parallel convolutions. Also, we observe that with more training episodes, the `topo` and `dfs-preorder` start to show fewer advantages as the number of times they find the best placement with the lowest training time decreases.

We also find out that the diameter of the input DAG affects which graph traversal order is performing the best in the `cifar10` dataset. With a smaller diameter (e.g., diameters smaller than 100), the `dfs` family (`dfs-preorder` and `dfs-postorder`) performs the best. With a larger diameter (e.g., diameters larger than 100), the `topo` family (`topo` and `reversed-topo`) tends to find better placements. This could be explained by the fact that the `dfs` family forms longer sequences of consecutive nodes on the diameter with a larger diameter. This can be hard for the RL agent to learn the placement of sibling nodes in the DAG as they are far away from each other in the sequence. This might require the RL agent to learn placement collocation of sibling nodes far away from each other.

4.4 Discussion and Guidelines

In the previous subsection, we show that graph traversal orders affect the training time of parallelized DL models. It means that a proper graph traversal order can help the RL agent to find better placements for DL models to reduce DL models' training time. Nevertheless, in Table 5 we show that if we give enough budget (time) to the RL agent to find the placements, then different graph traversal orders lead to placements of similar qualities (i.e., similar DL training time). Table 5 compares the ratio of the DL training speed found by different graph traversal orders versus the fastest training speed at episode 49, which is enough training budget based on our empirical study. For example, for `cifar10` dataset with eight devices, the placements found using `dfs-postorder` have an average training speed of one epoch per time unit, while the placements found using `dfs-preorder` have an average training speed of 0.97 epoch per time unit. The values of each row are normalized by the fastest training speed (i.e., the values are between zero and one, where one is the fastest). However, the efficiencies are different when the training budget is limited in real-world settings where larger DL models take much longer to find placement. Furthermore, the time saved for

Table 5. The placement training time comparison between different graph traversal orders. Each row shows a comparison of the average training speed of the DL model according to the placement found by different graph traversal orders at episode 49.

episode 49		Graph Traversal Order					
dataset	**#dev**	`lexico`	`topo`	`dfs-preorder`	`reversed-topo`	`dfs-postorder`	`bfs`
`nmt`	3	0.931	0.922	0.948	1.000	0.963	0.978
	5	0.869	0.863	0.929	1.000	0.954	0.966
	8	0.849	0.831	0.953	1.000	0.960	0.972
`ptb`	3	0.969	0.980	0.990	0.986	0.981	1.000
	5	0.953	0.962	0.969	0.976	0.974	1.000
	8	0.953	0.953	0.977	0.983	0.983	1.000
`cifar10`	3	0.975	0.982	1.000	0.997	0.990	0.978
	5	0.976	0.989	0.999	1.000	0.994	0.974
	8	0.945	0.970	0.970	0.995	1.000	0.929

training DL models with a 5% speedup is still not negligible when the DL model would take weeks, if not months, to train on a GPU cluster.

Identifying the proper graph traversal order for a DAG can improve the training efficiency that leads to better placements with lower training time on distributed hardware. However, finding the optimal graph traversal order for a given DL model is not an easy task as many factors are involved in the process, e.g., the topology of the DAG, the ratio of computation, and the communication during training. Although one cannot always quickly find the best graph traversal order for a DAG, we can still provide some guidelines based on our experience.

In general, it is good to start experiments with graph traversal orders that traverse the nodes in a DAG in a backward fashion, i.e., start from the nodes in the final layer of the graph, gradually go through the nodes in the previous layers, and finish with the nodes in the first layer of the model. For example, when using the `reversed-topo` order, the RL agent in the device placement method can first learn the placement of the nodes in the last layers and then on the nodes that are input to nodes that the RL agent already find placements for. By starting from backward, the RL agent can learn to better collocate parent and children nodes.

If a DAG has a large diameter and a large number of nodes or groups of nodes, then a graph traversal order that can put sibling nodes near each other in the one-dimensional sequence is a better candidate for the optimal graph traversal order. For example, when facing a large DL model with more than 200 nodes, the `bfs` order can put sibling nodes close to each other in the one-dimensional sequence. Thus, the RL agent can learn to better place the sibling nodes consecutively, instead of having to remember the placements of sibling nodes that are far away from each other in a long sequence.

In the context of the ExtremeEarth project [6,12,13], different types of models are used to provide EO products. While hyperparameter tuning [18] and ablation studies [28] can help to improve model performance, identifying proper graph traversal order can improve the model parallel training performance. For

example, for Synthetic Aperture Radar (SAR) image classification [10, 11], the `reversed-topo` and `dfs-postorder` would be good traversal orders to start the experiment, as the models are similar to those in `cifar10` model datasets. For sequence classification tasks [24], the `bfs` order would be a good traversal order to start with, as they are sequence to sequence models, which are similar to those in the `ptb` model datasets. The `bfs` order can help the RL agent to collocate better the placements of sibling operations in the DL model.

5 Related Work

In this section, we discuss related work in device placement. We start with a general overview of methods in device placement and then focus on those more related to graph traversal orders.

The first effort in device placement uses partition methods. For example, Mayer et al. [17] use a two-step approach that first partition the computation graph and then locally schedule the operations in each partition on each device. In another work, Tanaka et al. [31] present RaNNC that uses a three-step approach to partition the computation graph by first distinguishing atomic components, then coarsely partitioning the graph, and finally searching for the combination of the coarse partitions to find the final partition.

Mirhoseini et al. [19, 21] are the first to use RL approaches for device placement. In [19], they present HDP to find the placement by jointly learning the grouping operations in a computation graph and placement of the groups. This way, they improve the device placement time. Similarly, Lan et al. [16] present EAGLE that combines the automatic grouping of operations and finding placement for each group that improves the speed of finding better placements.

For making the device placement methods more generalized, Addanki et al. [1] introduce Placeto, which uses a graph embedding method for representing nodes in the graph and placing them iteratively. GPD [38], which is a single-shot device placement method, and Mars [15], which uses self-supervised pre-training to capture the topological relations between nodes in the computation graph, are other examples of generalized device placement methods. Gao et al. [4] present Spotlight that improves the training time of device placement methods by introducing a new RL algorithm based on proximal policy optimization. Later, they introduce Post [3] that further improves training efficiency by combining cross-entropy minimization and proximal policy optimization.

Some of the previous works study the relationship between graph traversal orders and the training time of the final placement found. For example, HDP [19] randomizes the order of predicting placements for each group of operations in a DL model. The authors find that the difference between the fastest and slowest placements is less than 7% in 10 experiments. Placeto [1] uses GNN to eliminate the need to assign indices when embedding graph features. Experiment results show that the predicted placement of Placeto is more robust to graph traversal orders than the RNN-based approaches. REGAL [23] uses topological order to convert a graph into a sequence. Mitropolitsky et al. [22] study

how different graph embedding techniques affect the execution time of the final placement and show that position-aware graph embedding improves the training time of the placement found compared to Placeto-GNN [1] and GraphSAGE [7]. GPD [38] removes the positional embedding in the transformer model to prevent overfitting.

Some work in other domains also studies graph traversal orders. In chip placement, Mirhoseini et al. [20] find that topological order can help the RL agent to place connected nodes close to each other. In the domain of generating graphs with DL models, GraphRNN [37] uses BFS order for graph generation to reduce the complexity of learning over all possible node sequences. The only possible edges for a new node are those connecting to nodes in the "frontier" of the BFS order. To the best of our knowledge, our work is the first to study how the graph traversal orders affect device placement training efficiency in device placement.

6 Conclusion

This work studies the impact of graph traversal orders on device placement for accelerating model parallel deep learning training. We empirically show that graph traversal orders affect the device placement and, consequently, the training time of deep learning models. A device placement method can learn more efficiently during the training process by finding placement strategies with lower training time faster when given a proper graph traversal order. Specifically, we find that traversing the computation graph from the nodes in the output layer of a deep learning model to the nodes in the input layer helps device placement methods find good placements efficiently. Moreover, we observe that for larger computation graphs, traversing orders that can better collocate sibling nodes, e.g., breadth-first search, in the traversal sequence is more efficient than its depth-first counterparts. We also provide practical guidelines on choosing traversal orders for device placement.

We believe that our study can help researchers and practitioners better understand the relationship between types of network and graph traversal orders. Several potential extensions and improvements to this work exist, including jointly learning graph traversal orders, graph embedding, and the policy network in the RL agent. Another possible direction is to study graph traversal orders based on the graph structures and features of individual nodes (e.g., input and output size and computation intensity of the given node). We can also investigate other optimization techniques, such as constraint programming, to solve device placement in future work.

Acknowledgements This work was supported by the ExtremeEarth project funded by European Union's Horizon 2020 Research and Innovation Programme under Grant agreement no. 825258.

References

1. Addanki, R., Bojja Venkatakrishnan, S., Gupta, S., Mao, H., Alizadeh, M.: Placeto: Learning generalizable device placement algorithms for distributed machine learning. Advances in Neural Information Processing Systems 32 (NIPS 2019) (2019)
2. Brown, T.B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al.: Language models are few-shot learners. arXiv preprint arXiv:2005.14165 (2020)
3. Gao, Y., Chen, L., Li, B.: Post: Device placement with cross-entropy minimization and proximal policy optimization. In: Advances in Neural Information Processing Systems. pp. 9971–9980 (2018)
4. Gao, Y., Chen, L., Li, B.: Spotlight: Optimizing device placement for training deep neural networks. In: International Conference on Machine Learning. pp. 1676–1684 (2018)
5. Hagberg, A., Swart, P., S Chult, D.: Exploring network structure, dynamics, and function using networkx. Tech. rep., Los Alamos National Lab.(LANL), Los Alamos, NM (United States) (2008)
6. Hagos, D.H., Kakantousis, T., Vlassov, V., Sheikholeslami, S., Wang, T., Dowling, J., Paris, C., Marinelli, D., Weikmann, G., Bruzzone, L., et al.: Extremeearth meets satellite data from space. IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing **14**, 9038–9063 (2021)
7. Hamilton, W.L., Ying, R., Leskovec, J.: Inductive representation learning on large graphs. In: Proceedings of the 31st International Conference on Neural Information Processing Systems. pp. 1025–1035 (2017)
8. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural computation **9**(8), 1735–1780 (1997)
9. Kahn, A.B.: Topological sorting of large networks. Communications of the ACM **5**(11), 558–562 (1962)
10. Khaleghian, S., Kramer, T., Everett, A., Kiarbech, A., Hughes, N., Eltoft, T., Marinoni, A.: Synthetic aperture radar data analysis by deep learning for automatic sea ice classification. In: EUSAR 2021; 13th European Conference on Synthetic Aperture Radar. pp. 1–6. VDE (2021)
11. Khaleghian, S., Ullah, H., Kræmer, T., Hughes, N., Eltoft, T., Marinoni, A.: Sea ice classification of sar imagery based on convolution neural networks. Remote Sensing **13**(9), 1734 (2021)
12. Koubarakis, M., Bereta, K., Bilidas, D., Giannousis, K., Ioannidis, T., Pantazi, D.A., Stamoulis, G., Haridi, S., Vlassov, V., Bruzzone, L., et al.: From copernicus big data to extreme earth analytics. Open Proceedings pp. 690–693 (2019)
13. Koubarakis, M., Stamoulis, G., Bilidas, D., Ioannidis, T., Mandilaras, G., Pantazi, D.A., Papadakis, G., Vlassov, V., Payberah, A.H., Wang, T., et al.: Artificial intelligence and big data technologies for copernicus data: The extremeearth project. In: Proceedings of the 2021 conference on Big Data from Space. Publications Office of the European Union (2021)
14. Kreuzer, D., Beaini, D., Hamilton, W.L., Létourneau, V., Tossou, P.: Rethinking graph transformers with spectral attention. arXiv preprint arXiv:2106.03893 (2021)
15. Lan, H., Chen, L., Li, B.: Accelerated device placement optimization with contrastive learning. In: 50th International Conference on Parallel Processing. pp. 1–10 (2021)

16. Lan, H., Chen, L., Li, B.: Eagle: Expedited device placement with automatic grouping for large models. In: 2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS). pp. 599–608 (2021). https://doi.org/10.1109/IPDPS49936.2021.00068
17. Mayer, R., Mayer, C., Laich, L.: The tensorflow partitioning and scheduling problem: it's the critical path! In: Proceedings of the 1st Workshop on Distributed Infrastructures for Deep Learning. pp. 1–6 (2017)
18. Meister, M., Sheikholeslami, S., Payberah, A.H., Vlassov, V., Dowling, J.: Maggy: Scalable asynchronous parallel hyperparameter search. In: Proceedings of the 1st Workshop on Distributed Machine Learning. pp. 28–33 (2020)
19. Mirhoseini, A., Goldie, A., Pham, H., Steiner, B., Le, Q.V., Dean, J.: A hierarchical model for device placement. In: International Conference on Learning Representations (2018)
20. Mirhoseini, A., Goldie, A., Yazgan, M., Jiang, J.W., Songhori, E., Wang, S., Lee, Y.J., Johnson, E., Pathak, O., Nazi, A., et al.: A graph placement methodology for fast chip design. Nature **594**(7862), 207–212 (2021)
21. Mirhoseini, A., Pham, H., Le, Q.V., Steiner, B., Larsen, R., Zhou, Y., Kumar, N., Norouzi, M., Bengio, S., Dean, J.: Device placement optimization with reinforcement learning. In: International Conference on Machine Learning. pp. 2430–2439. PMLR (2017)
22. Mitropolitsky, M., Abbas, Z., Payberah, A.H.: Graph representation matters in device placement. In: Proceedings of the Workshop on Distributed Infrastructures for Deep Learning. pp. 1–6 (2020)
23. Paliwal, A., Gimeno, F., Nair, V., Li, Y., Lubin, M., Kohli, P., Vinyals, O.: Reinforced genetic algorithm learning for optimizing computation graphs. In: International Conference on Learning Representations (2020)
24. Paris, C., Weikmann, G., Bruzzone, L.: Monitoring of agricultural areas by using sentinel 2 image time series and deep learning techniques. In: Image and Signal Processing for Remote Sensing XXVI. vol. 11533, p. 115330K. International Society for Optics and Photonics (2020)
25. Pham, H., Guan, M., Zoph, B., Le, Q., Dean, J.: Efficient neural architecture search via parameters sharing. In: International Conference on Machine Learning. pp. 4095–4104. PMLR (2018)
26. Scarselli, F., Gori, M., Tsoi, A.C., Hagenbuchner, M., Monfardini, G.: The graph neural network model. IEEE transactions on neural networks **20**(1), 61–80 (2008)
27. Shallue, C.J., Lee, J., Antognini, J., Sohl-Dickstein, J., Frostig, R., Dahl, G.E.: Measuring the effects of data parallelism on neural network training. Journal of Machine Learning Research **20**(112), 1–49 (2019)
28. Sheikholeslami, S., Meister, M., Wang, T., Payberah, A.H., Vlassov, V., Dowling, J.: Autoablation: Automated parallel ablation studies for deep learning. In: Proceedings of the 1st Workshop on Machine Learning and Systems. pp. 55–61 (2021)
29. Shoeybi, M., Patwary, M., Puri, R., LeGresley, P., Casper, J., Catanzaro, B.: Megatron-lm: Training multi-billion parameter language models using model parallelism. arXiv preprint arXiv:1909.08053 (2019)
30. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A.: Going deeper with convolutions. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 1–9 (2015)
31. Tanaka, M., Taura, K., Hanawa, T., Torisawa, K.: Automatic graph partitioning for very large-scale deep learning. arXiv preprint arXiv:2103.16063 (2021)

32. Topping, J., Di Giovanni, F., Chamberlain, B.P., Dong, X., Bronstein, M.M.: Understanding over-squashing and bottlenecks on graphs via curvature. arXiv preprint arXiv:2111.14522 (2021)
33. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. In: Advances in neural information processing systems. pp. 5998–6008 (2017)
34. Williams, R.J.: Simple statistical gradient-following algorithms for connectionist reinforcement learning. Machine learning **8**(3), 229–256 (1992)
35. Wu, Y., Schuster, M., Chen, Z., Le, Q.V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., et al.: Google's neural machine translation system: Bridging the gap between human and machine translation. arXiv preprint arXiv:1609.08144 (2016)
36. Ying, C., Cai, T., Luo, S., Zheng, S., Ke, G., He, D., Shen, Y., Liu, T.Y.: Do transformers really perform bad for graph representation? arXiv preprint arXiv:2106.05234 (2021)
37. You, J., Ying, R., Ren, X., Hamilton, W., Leskovec, J.: Graphrnn: Generating realistic graphs with deep auto-regressive models. In: International conference on machine learning. pp. 5708–5717. PMLR (2018)
38. Zhou, Y., Roy, S., Abdolrashidi, A., Wong, D., Ma, P.C., Xu, Q., Zhong, M., Liu, H., Goldie, A., Mirhoseini, A., et al.: Gdp: Generalized device placement for dataflow graphs. arXiv preprint arXiv:1910.01578 (2019)
39. Zhou, Y., Roy, S., Abdolrashidi, A., Wong, D.L.K., Ma, P., Xu, Q., Mirhoseini, A., Laudon, J.: A single-shot generalized device placement for large dataflow graphs. IEEE Micro **40**(5), 26–36 (2020)
40. Zhu, X.X., Tuia, D., Mou, L., Xia, G.S., Zhang, L., Xu, F., Fraundorfer, F.: Deep learning in remote sensing: A comprehensive review and list of resources. IEEE Geoscience and Remote Sensing Magazine **5**(4), 8–36 (2017)