Amazon SageMaker Model Parallelism: A General and Flexible Framework for Large Model Training

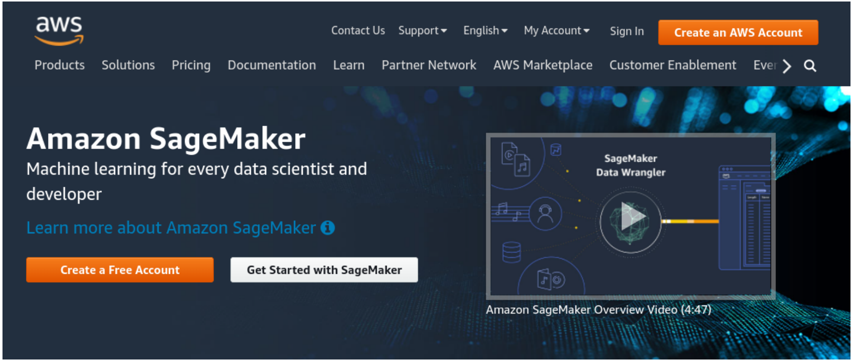
Can Karakus[1] Rahul Huilgol[1] Fei Wu[1] Anirudh Subramanian[1] Cade Daniel[1]
Derya Cavdar[1] Teng Xu[1] Haohan Chen[1] Arash Rahnama[1] Luis Quintela[1]

[1]Amazon

Presenter: Shiwei Zhang

Prelude

Amazon SageMaker



Amazon SageMaker

Machine learning for every data scientist and developer

Learn more about Amazon SageMaker

Amazon SageMaker Overview Video (4:47)

Content

- Design Principle and Overview
- Pipeline Parallelism
- Tensor Parallelism
- Experiments
- Summary

Design Principle and Framework Overview

Motivation

Existing systems are not flexible enough to handle:

- architectures that do not consist of a single transformer encoder or large non-transformer architectures,
- architectures that do not consist of a consecutive sequence of identical layers,
- a single large component in an otherwise small model,
- architectures that make extensive use of module/parameter re-use,
- scripts with conditional execution flows,
- and non-conventional execution pattern such as mixture-of-experts layers.

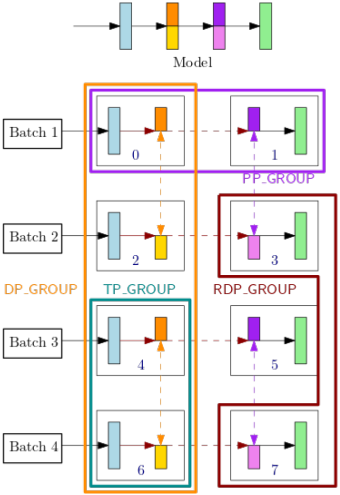
Design Principles

- Do not abstract away the training step
- Preserve framework features and characteristics
- Do not limit to specific architectures

Framework Overview

The library features pipeline, tensor, and data parallelism, controlled by the three groups.

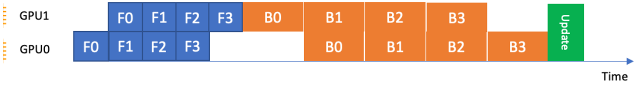
The entry function (training loop) is wrapped in a `smp.step` decorator to activate this library.

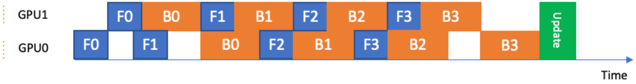


Pipeline Parallelism

Background

In pipeline parallelism, the model is partitioned into **stages**. Each stage is put on one device. Multiple microbatches are running concurrently in a pipeline.

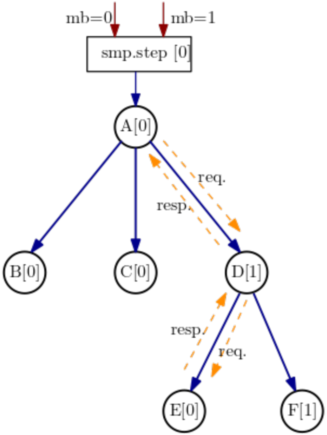




Tree Structure and Message Passing Workflow

The model is viewed as a hierarchy of modules based on PyTorch class definition.

Each pipeline stage runs an execution server, which is a Python thread that listen requests from incoming execution requests



Partitioning Algorithm

Problem Definition

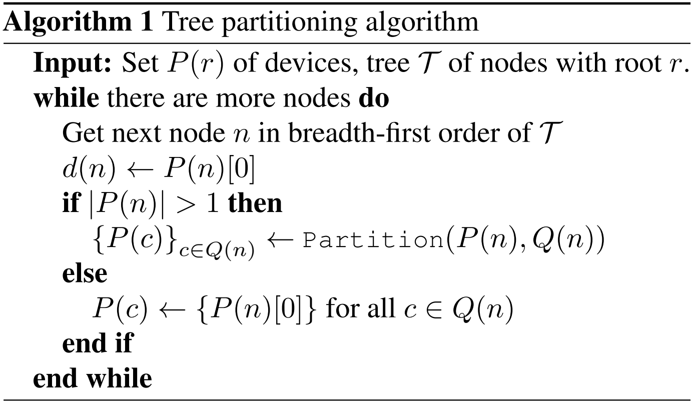
For a tree $\mathcal{T} = (\mathcal{V}, \mathcal{E})$, we define $Q(n)$ as the set of children of a node $n \in \mathcal{V}$. A cost $c(n)$ is associated with each node n representing the memory and computation cost of the subtree.

The goal of the algorithm is to find the device assignment $d(n)$ for each node n.

Virtual devices: in this part, “device” stands for the unit of pipeline groups, which may contain multiple GPUs.

Module Nodes: A Node in the tree V may contain multiple modules if they share the same parameter. In this case, the edges are arbitrarily pruned to keep it a tree.

Main loop



Algorithm 1 Tree partitioning algorithm

```
Input: Set $P(r)$ of devices, tree $\mathcal{T}$ of nodes with root $r$.
while there are more nodes do
    Get next node $n$ in breadth-first order of $\mathcal{T}$
    $d(n) \leftarrow P(n)[0]$
    if $|P(n)| > 1$ then
        $\{P(c)\}_{c \in Q(n)} \leftarrow \texttt{Partition}(P(n), Q(n))$
    else
        $P(c) \leftarrow \{P(n)[0]\}$ for all $c \in Q(n)$
    end if
end while
```

Cost function

$$\bar{C}(m) = \alpha w(m) + (1-\alpha)\psi(m)$$

where $\bar{C}(m)$ is the unnormalized cost, $w(m)$ is the memory cost, and $\psi(m)$ is the computation cost of module m.

The cost of a Module Node n is defined recursively as the sum of the costs of the set of modules $M(n)$ it contains and the costs of its children $Q(n)$

$$C(n) = \sum_{m \in M(n)} \bar{C}(m) + \sum_{p \in Q(n)} C(p)$$

The final cost is normalized by the cost of the root r

$$c(n) = \frac{C(n)}{C(r)}$$

Segmentation

To assign the set of devices $P(n)$ to each child node in $Q(n)$, we first partition the sequence of $Q(n)$ into l segments such that

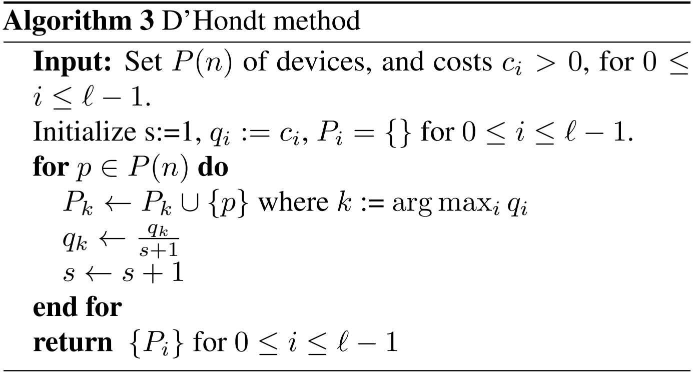
$$\min_{\mathcal{P}} \omega(\mathcal{P}) = \min_{\mathcal{P}} \max_{S \in P} \sum_{p \in S} c(p)$$

This can be solved with dynamic programming using the recursion:

$$c(k, i) = \min_{j \leq i} \max\{c(k-1, j), \sum_{q \in Q(n,j)} c(q)\}$$

where $c(k, i)$ is the partition cost $\mathcal{P}$ achieved in partitioning the first i elements of $Q(n)$ into k partitions, and $Q(n, j)$ represents the sub-sequence of $Q(n)$ from element j onwards.

Device Allocation



Algorithm 3 D'Hondt method

```
Input: Set $P(n)$ of devices, and costs $c_i > 0$, for $0 \leq i \leq \ell - 1$.
Initialize s:=1, $q_i := c_i$, $P_i = \{\}$ for $0 \leq i \leq \ell - 1$.
for $p \in P(n)$ do
    $P_k \leftarrow P_k \cup \{p\}$ where $k :=$ arg max$_i$ $q_i$
    $q_k \leftarrow \frac{q_k}{s+1}$
    $s \leftarrow s + 1$
end for
return $\{P_i\}$ for $0 \leq i \leq \ell - 1$
```

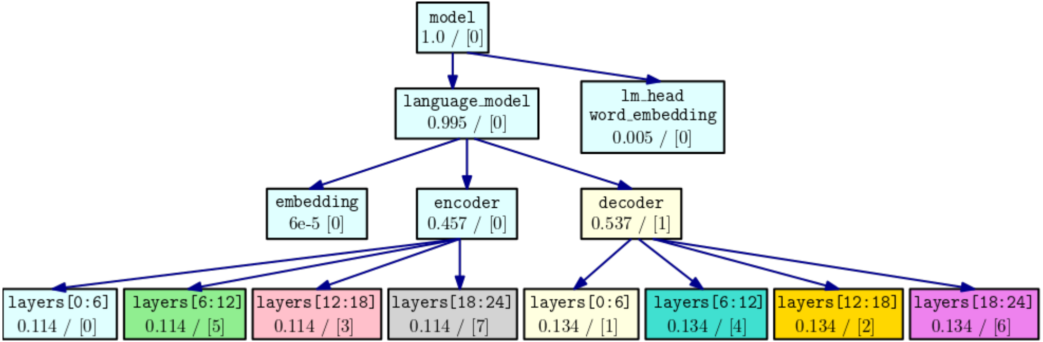
Recursive partitioning

At the end of D'Hondt allocation, there are three possible scenarios for each segment:

- No device is assigned to it. This segment will be put on the same device as the parent node.
- One device is assigned to the segment. The segment and all its subtrees will be placed on the device.
- Multiple devices are assigned to the segment. If this segment has only one node, all devices are assigned to it (it will be revisited in Algorithm 1). Otherwise, the segment is further divided into smaller segments recursively.

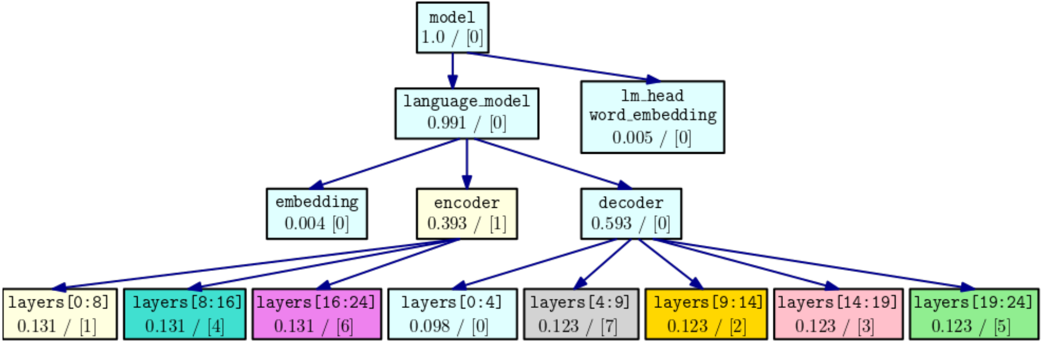
Example Partitioning

Auto-partition decision over T5-11B with $\alpha = 1.0$



Another Example Partitioning

Auto-partition decision over T5-11B with $\alpha = 0.2$



Tensor Parallelism

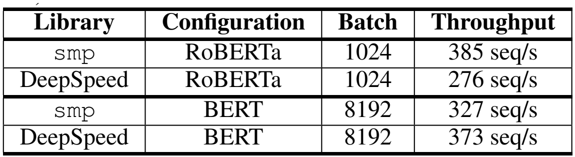
Tensor Parallelism

SageMaker achieves tensor parallelism by providing drop-in replacement layers like `DistributedLiner`, `DistributedEmbedding`, `DistributedTransformer`, etc.

Experiments

10-billion parameter RoBERTa and BERT

- Hardware: 16 nodes, each equipped with 8 A100.
- Models: RoBERTa/BERT, 10B parameters
- Baselines: DeepSpeed with ZeRO stage 2



Library	Configuration	Batch	Throughput
smp	RoBERTa	1024	385 seq/s
DeepSpeed	RoBERTa	1024	276 seq/s
smp	BERT	8192	327 seq/s
DeepSpeed	BERT	8192	373 seq/s

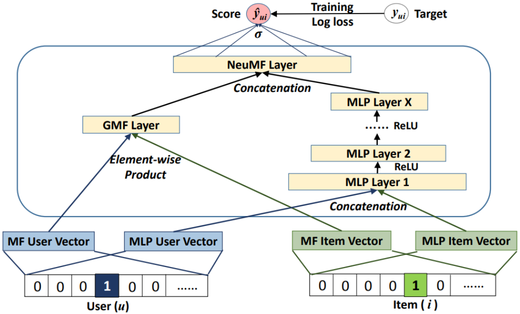
175-billion parameter GPT-3

A GPT-3 model with 175-billion parameters and a sequence length of 2048 is trained on 120 nodes for a total of 960 A100 GPUs. The pipeline parallelism degree is 6 and tensor parallelism degree is 8.

Neural collaborative filtering

- Hardware: 4 nodes, each equipped with 8 A100.
- Model: NCF
- Baseline: Pure tensor-parallelism

Setting	Dataset	Throughput
smp	1	107656 samples/s
Baseline	1	43078 samples/s
smp	2	69298 samples/s
Baseline	2	36723 samples/s



Summary

Conclusion

Strength

- Tree structure and message passing workflow.
- Automatic pipeline parallelism based on dynamic programming and D'Hondt method.

Limitation

- The proposed methods do not solve the problems listed in motivation.
- The experiment results are not very good.
- There is no analysis (optimality, complexity, etc.) for the proposed methods.

Takeaways

- In addition to the linearly-connected layers and DAG structures, we can also view a model as a tree.
- The message-passing architecture may provide new challenges and opportunities.

Thank you!