Accelerating Large-Scale Distributed Neural Network Training with SPMD Parallelism

Shiwei Zhang[1] Lansong Diao[2] Chuan Wu[1] Siyu Wang[2] Wei Lin[2]

[1]The University of Hong Kong
[2]Alibaba Group

ACM SoCC 2022

香 港 大 學
THE UNIVERSITY OF HONG KONG

Alibaba

Deep neural networks (DNN), especially Transformer-based models with **Mixture-of-Expert (MoE)** layers, have become so large that distributed training is necessary.

Communication overhead is a major problem in distributed DNN training. Using **TPUs** with fast device-to-device links, communication can take up to **11%** of training time. On **GPU clouds** with Ethernet connection, communication can take more than **60%** of training time.

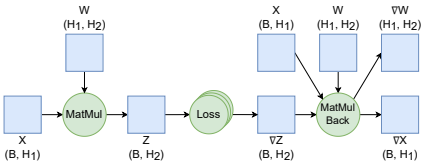
We present our system, HiDup, that mitigates the communication overhead by **computation-communication overlapping** and **overlapping-aware sharding strategy**.

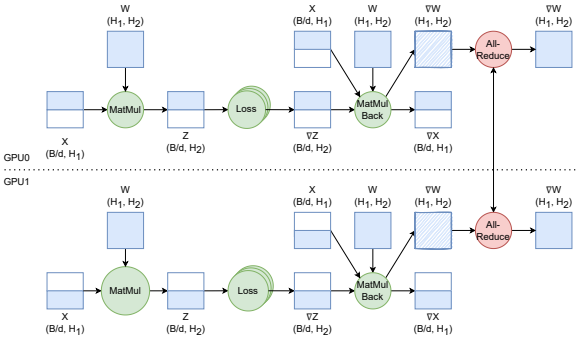
Content

Background and Motivation

Data Parallelism

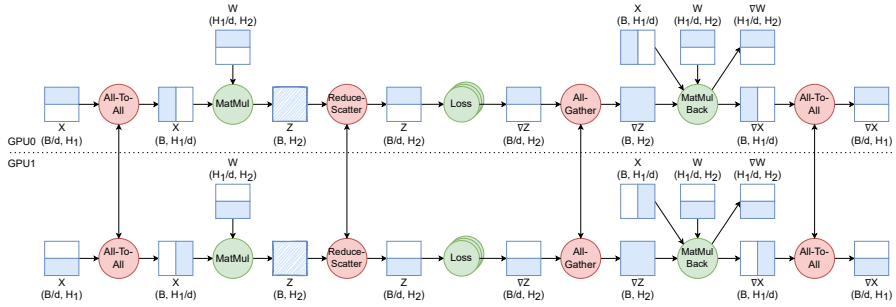
With data parallelism (DP), a minibatch of training data is split along the batch size dimension and each worker processes a slice independently. `All-Reduce` is used to synchonize the grandients at the end of each iteration.



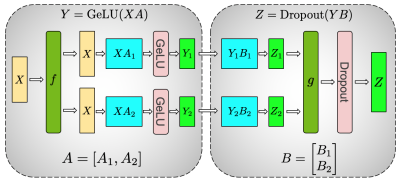


SPMD Parallelism

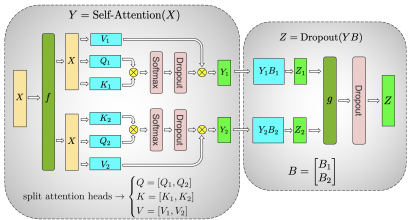
DP does not support models that exceed the memory of a single device. SPMD parallelism is a more general method that allows sharding along any dimension of any tensor. Collective communication is needed when the sharding methods of two tensors are incompatible.



SPMD Parallelism Examples

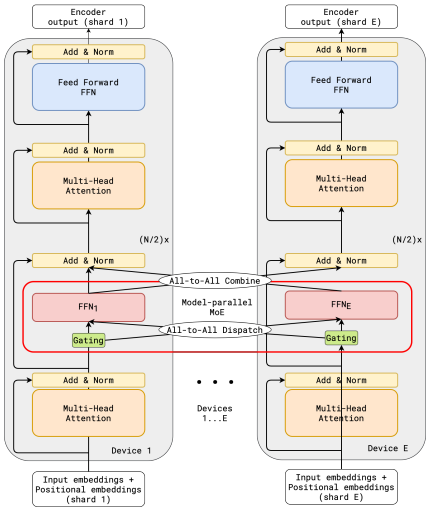


(a) MLP



(b) Self-Attention

Source: Shoeybi, M., et al. "Megatron-LM" (2019).

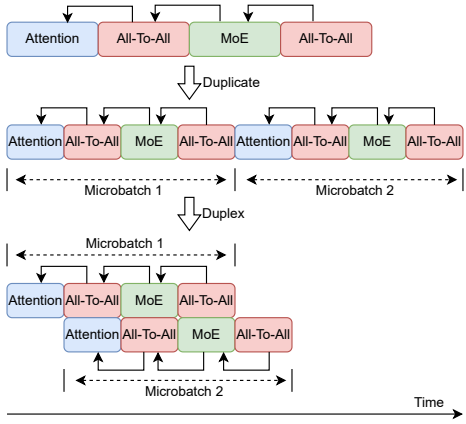


Source: Lepikhin, D., et al. "GShard" (2020).

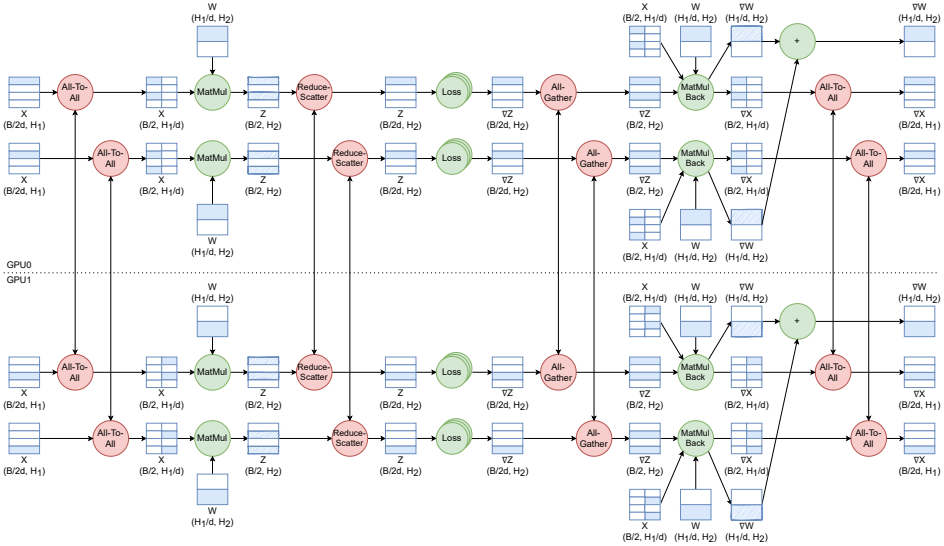
Duplex: Enable Computation-Communication Overlapping with SPMD Parallelism

Duplex

Inspired by gradient accumulation, we split the input data on each worker into two microbatches **after applying SPMD parallelism**, and scheduling the two microbatches into a pipeline such that the computation of one microbatch overlaps with the communication of the other microbatch.



Duplex Example



Duplex

- **Accuracy**: Only floating-point arithmetic errors.
- **Memory Usage**: No increase.
- **GPU Utilization**: Slightly reduced due to smaller tensor sizes.
- **Overheads**: CUDA synchonization, gradient aggregation, memory bus contention, interference between computation and communication.
- **Speed-up**: Up to 100% in ideal case.

Duplex-aware Sharding Strategy

Motivation

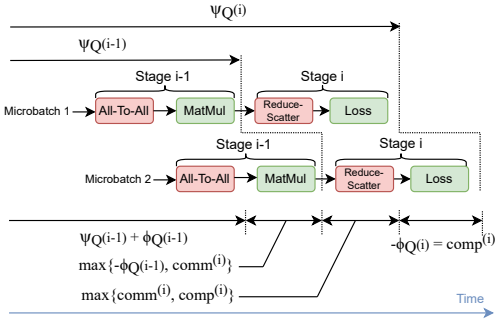
With the fast emergence of new DNN models, manually designing SPMD strategies for each model is manpower intensive and time-consuming. Further, the strategy may not always be optimal on different clusters.

Existing studies search for sharding strategy that **minimizes communication volume**. This strategy may not be optimal when used together with Duplex.

Sharding Strategy

- **Objective**: Find the sharding methods for all operators that minimizes the training time with computation-communication overlapping.
- **Basic Idea**: Using dynamic programming to incrementally search for the best strategies of a subgraph.
- **Challenge**: The best strategy of a subgraph may not be optimal for the complete graph due to computation-communication overlapping.
- **Solution**: We propose a stage-based cost model and track two costs associated with a search state.

Sharding Strategy



More details in the paper.

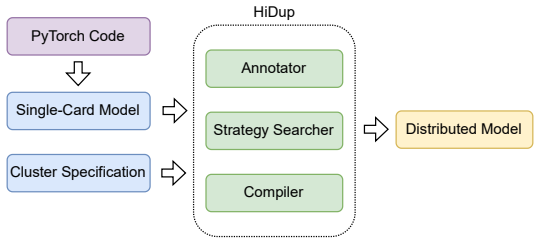
Input: Computation graph G
Output: Optimal SPMD strategy Q^*

Initialize P with an empty strategy $Q_\emptyset$
for $C = C_0$ **to** C_n **do**
  for $Q \in P$ where $C \in \text{state}(Q)$ **do**
    for each (O, S) that can be appended to Q **do**
      $Q\prime \leftarrow Q \oplus (O, S)$
      if $\exists Q_p \in P$ s.t. $\text{state}(Q_p) = \text{state}(Q')$ and Q' is dominated by Q_p **then**
        continue
      end if
      for $Q_p \in P$ where $\text{state}(Q_p) = \text{state}(Q')$ **do**
        if Q_p is dominated by $Q\prime$ **then**
          remove Q_p from P
        end if
      end for
      append $Q\prime$ into P
    end for
  end for
end for

Implementation

Implementation

We implement HiDup as a graph transformation module on PyTorch. It takes as input a single-card model (as PyTorch fx graph) and the cluster specification, producing a modified graph that runs on all workers.



Implementation

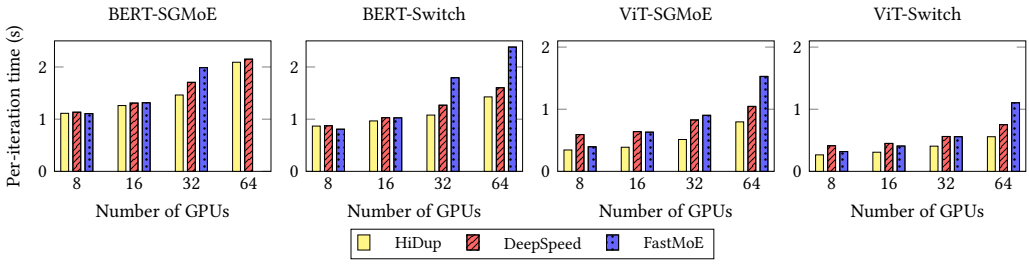
- **Annotator**: Label the possible sharding methods for each operator, infer the tensor sizes, and estimate the FLOPs.
- **Strategy Searcher**: Take annotated graph as input and search for the optimal sharding strategy.
- **Compiler**: Modify the graph according to the strategy and applies Duplex.

Evaluation

Experimental Setup

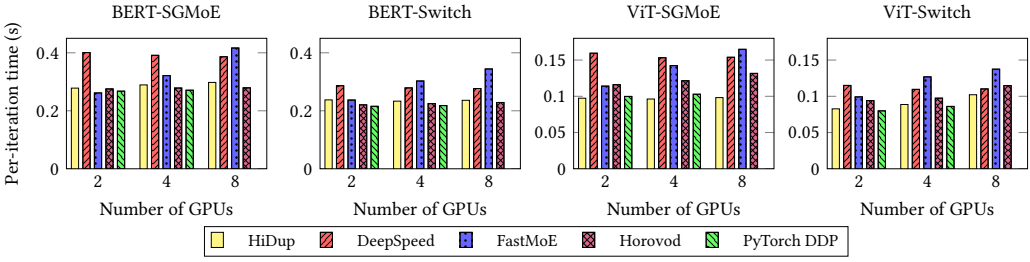
- **Testbed**: 8 machines on public cloud, each with 8 V100 GPUs and NVLink, connected by 10Gbps network.
- **Benchmarks**: BERT (language modeling) and ViT (image classification), with two variants of MoE layers, SGMoE and Switch.
- **Baselines**: DeepSpeed, FastMoE, PyTorch DDP, and Horovod.

Per-iteration training time



HiDup outperforms baselines when scaling up, because the collective communication becomes slower with more cards and HiDup can mitigate the increased communication overhead with our Duplex design.

Single Machine Performance



Pure-DP methods perform well with high bandwidth. HiDup automatically identifies similar strategies and achieves comparable performance despite of the additional overheads introduced by our Duplex design.

Per-iteraion time breakdown

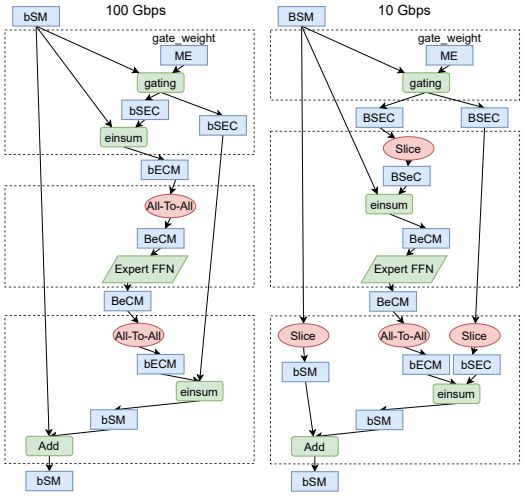
	Single Machine			Two Machines (100Gbps)			Two Machines (30Gbps)		
System	HiDup	DeepSpeed	FastMoE	HiDup	DeepSpeed	FastMoE	HiDup	DeepSpeed	FastMoE
Wall time (s)	0.2972	0.3988	0.3492	0.3985	0.4993	0.6971	0.6657	0.8251	0.7563
Total time (s)	0.3245	0.3813	0.3375	0.4611	0.4827	0.6857	0.7453	0.8055	0.7417
Computation (s)	0.2948	0.3531	0.2182	0.3088	0.3479	0.2252	0.3021	0.3469	0.2300
Communication (s)	0.0297	0.0282	0.1193	0.1523	0.1347	0.4605	0.4432	0.4585	0.5117
Overlapping ratio	91.92%	0	0	41.10%	0	0	26.35%	0	0

HiDup's total time is similar to the baselines, but it achieves shorter wall time by overlapping computation and communication.

SPMD Strategy

HiDup generates different strategies for the same model on different clusters.

It automatically identifies expert-designed strategies for common models.



Conclusion

Conclusion

- We propose Duplex that enables computation-communication overlapping with SPMD parallelism.
- We design a Duplex-aware sharding strategy search algorithm.
- We explore graph transformation on PyTorch and implement HiDup based on PyTorch fx.

Thank you!

Email: swzhang@cs.hku.hk