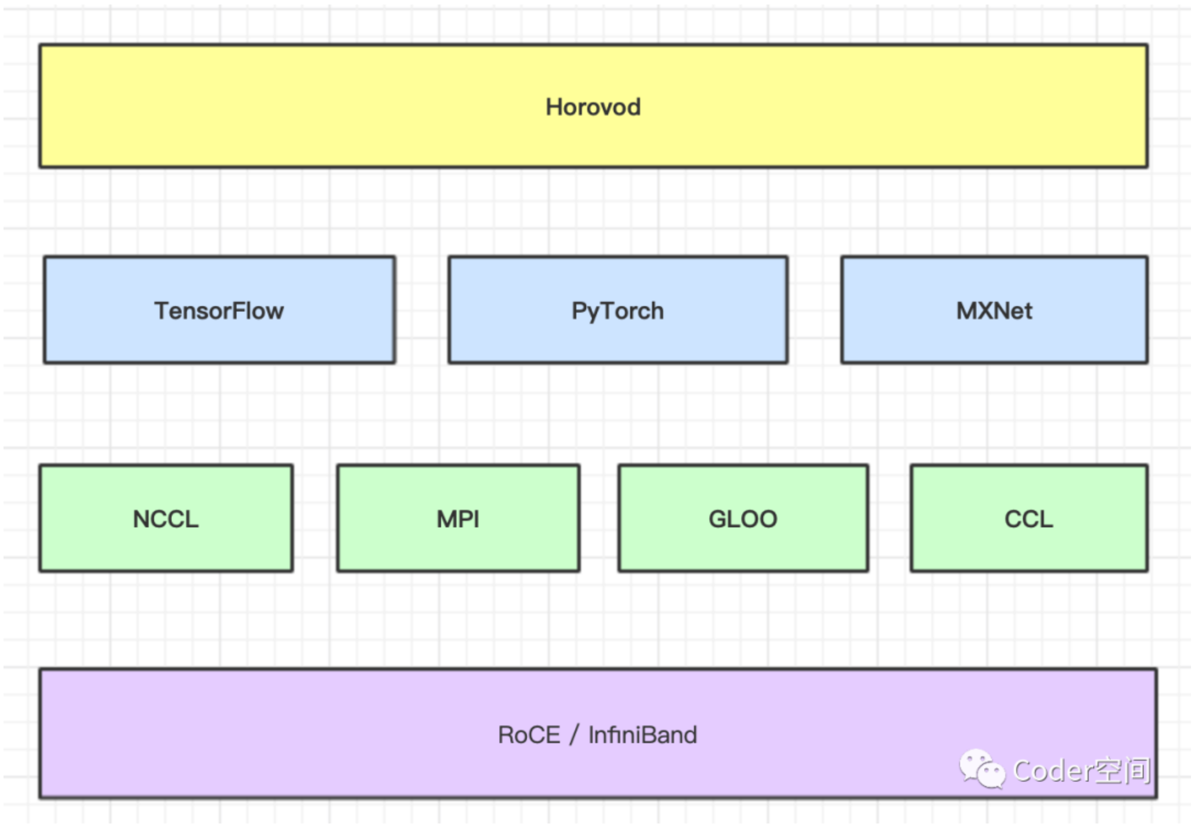
Accelerating Collective Communication in Data Parallel Training across Deep Learning Frameworks

introduction && motivation

Horovod

Horovod是一个基于数据并行的分布式训练框架

- 基于ring allreduce
- 支持TensorFlow，Pytorch等多种深度学习框架



motivation

1. Coordinator-worker的workload随着node的增加而增加，导致Horovod扩展性较差

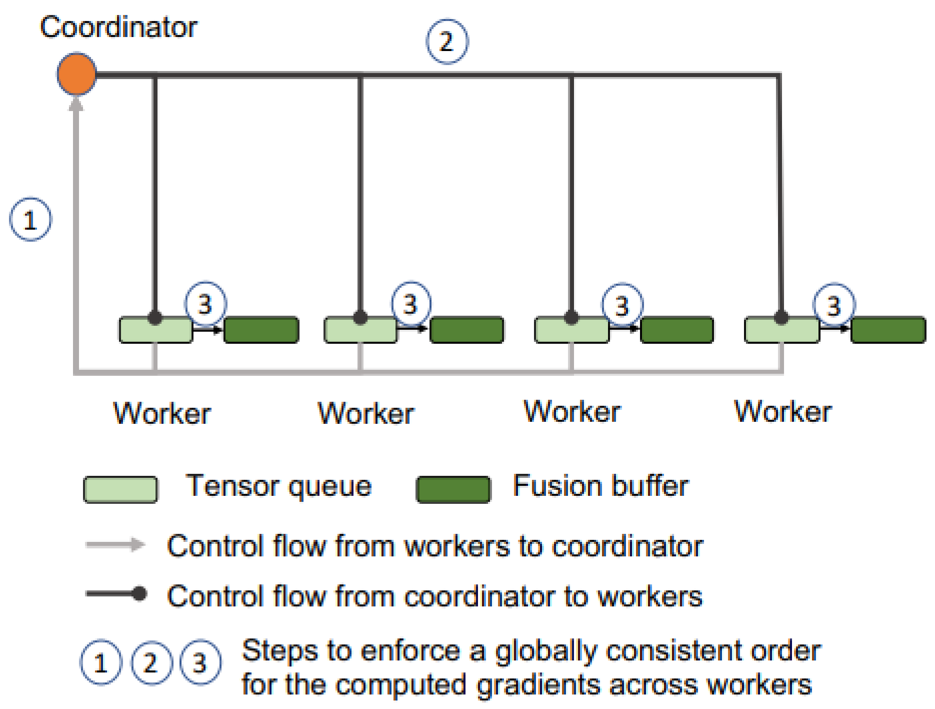

Coordinator
2
1
3
3
3
3
Worker
Worker
Worker
Worker
Tensor queue
Fusion buffer
Control flow from workers to coordinator
Control flow from coordinator to workers
1 2 3 Steps to enforce a globally consistent order for the computed gradients across workers

Figure 1: Coordinator-worker control model in Horovod's original design. The coordination progresses in three steps (see details in §2.2.2): First, the coordinator gathers the lists of requests from all workers; Second, the coordinator processes the request lists, and then generates and broadcasts a response list when observing one or more common requests from all workers; Third, after receiving the response list, each worker proceeds to execute collective operations.

> One observation from this analogy is that Horovod's design is inherently unaware of any aspects of DL training, in particular that in typical DL workloads, a fixed set of gradient tensors will be repeatedly AllReduced during the course of a training run (discussed in §2.1)

2. Horovod的buffering mechanism无法可靠的生成最优的缓存大小以有效利用带宽

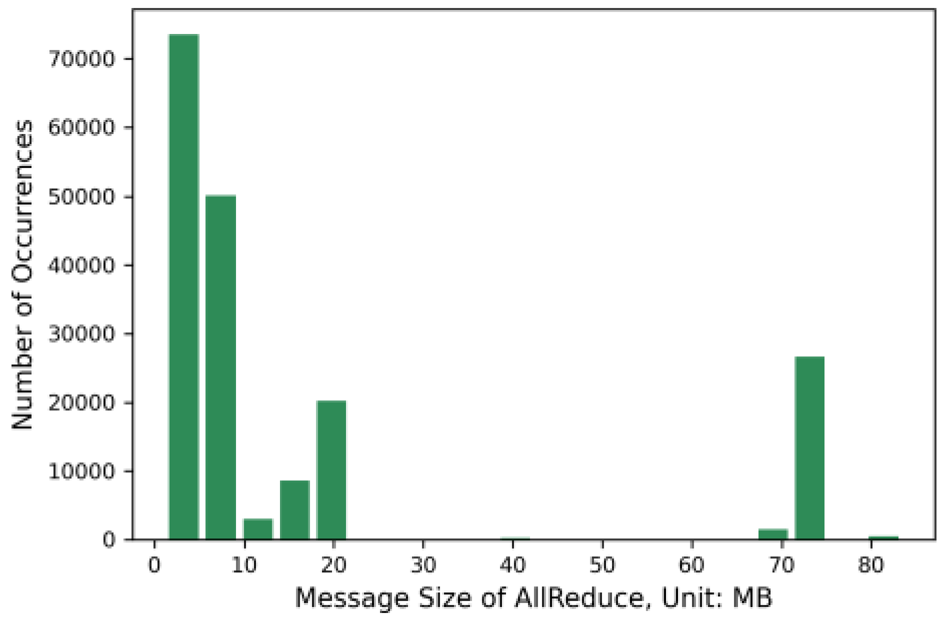


Figure 2: Histogram of AllReduce message size in Horovod's original design of Tensor Fusion. We present the results of a training run of ResNet50 with 96 GPUs on Summit (§4.1).

> After we analyzed the timelines of a few models, we noticed that those with a large amount of tensors, such as ResNet-101, tended to have many tiny allreduce operations. As noted earlier, ring-allreduce utilizes the network in an optimal way if the tensors are large enough, but does not work as efficiently or quickly if they are very small

Boosting Collective Communication in DNN Training with Caching and Grouping

In particular, we develop a light weight decentralized coordination strategy by utilizing a response cache.

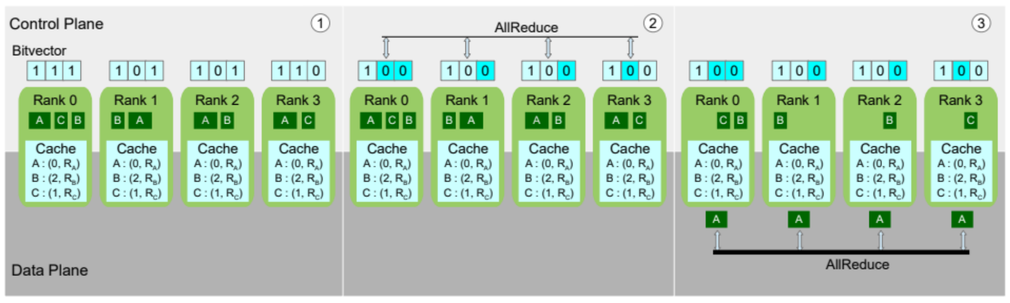


Figure 4: Illustration of AllReduce with Caching. We depict an example with 4 workers (0, 1, 2, 3) reducing 3 tensors (A, B, C). The strategy works in three steps: 1. Each worker populates a *bitvector*, setting bits according to entries in the *response cache* and the pending tensors in their local queues. 2. Workers synchronize the bitvectors via a global set intersection to identify common pending tensors. In this example, the bit associated with tensor A is shown as common across the workers. 3. Tensor A is sent to the data plane for AllReduce. When the AllReduce operation is done, Tensor A is removed from the queues on all workers.

1. 第一个cycle，每个worker执行逻辑与原来的版本保持一致，从本地的TensorFlow queue中获取准备好的request，生成RequestList.
2. 检查RequestList中的request是否已有相应的response cache.如果是的话,将相应的entry 在cache中的index加入到一个set:CacheBits.否则，设置一个标记表示该request已经准备好，但是还没有response cache
3. 每个worker根据CacheBits中的值填充比特向量BitVector。然后对worker的BitVector通过一个按位与操作执行allreduce，结果为一个global BitVectorg.

4. 每个worker解析global BitVector 以形成global CacheBits
5. 检查RequestList中的request。如果有对应的cache但是不在global CacheBits表示，还有worker没有准备好，下一轮再进行检查，如果有对应的cache且在 global CacheBits中，表示所有worker都已经准备好进行communication。将response cache取出加入ResponseList。如果没有对应的cache,则加入uncached request list,需要通过coordinator处理
6. 如果uncached request list为空，表明所有的request对应的response都已被cache，coordinator-worker process被跳过，worker直接处理ResponseList.否则通过coordinator处理uncached request list

GROUP

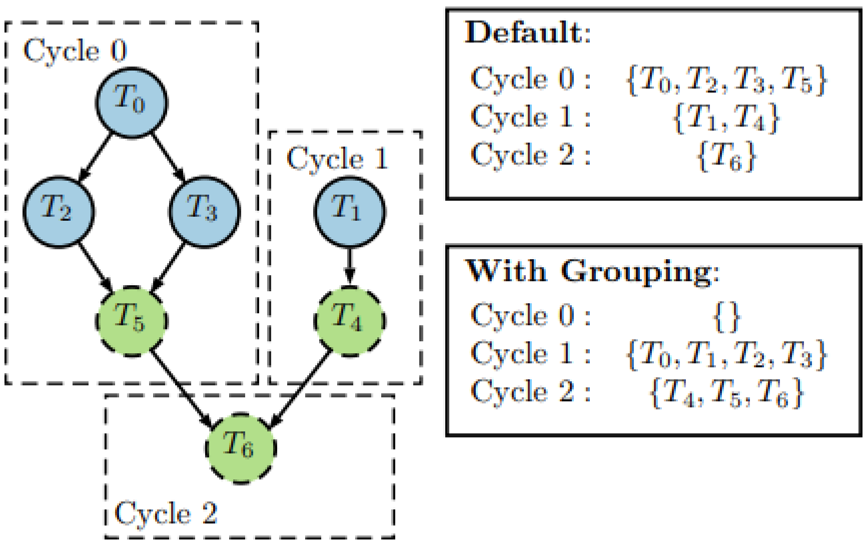


Figure 5: Illustration of Grouping. A task graph with nodes that generate requests T_n is depicted on the left, with the dashed boxes indicating requests visible to Horovod at 3 subsequent cycles. The nodes are colored to depict assignment to two groups (blue/solid borders and green/dashed borders). By default, a worker will submit all requests observed in a cycle to be processed/executed which can yield unbalanced sets of requests. With grouping enforced, requests are only submitted when complete groups are available.

> Collective requests submitted within a group are treated as a single request in Horovod's control plane; that is, no request in the group is considered ready for the data plane until all requests in the group are submitted

Experiment

Evaluations on Horovod's Strategies

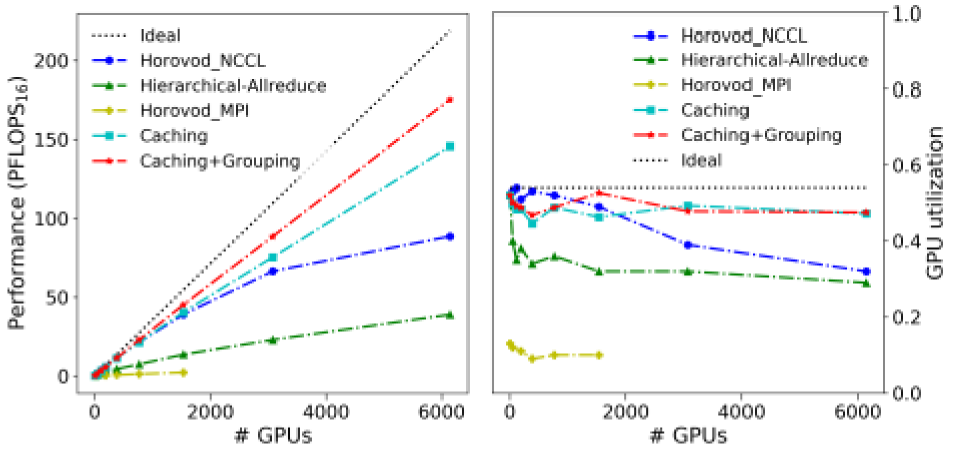


Figure 6: Performance and GPU utilization of Horovod's strategies. We compare our new techniques to the existing Horovod implementations using STEMDL (see Figure 3).

Evaluations across Frameworks and Communication Libraries

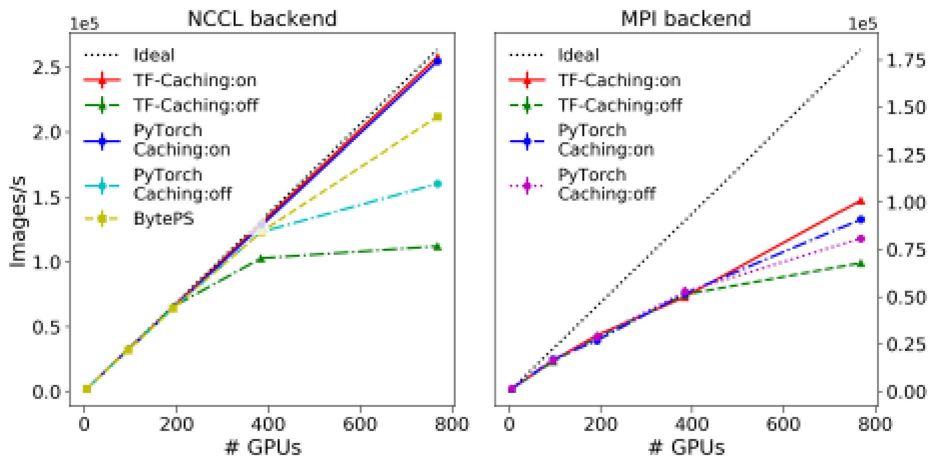


Figure 7: Performance of caching on ResNet50. We evaluate Horovod with caching enabled and disabled with both NCCL (left) and MPI (right) backends, and also compare the results to the performance of BytePS with NCCL (left).

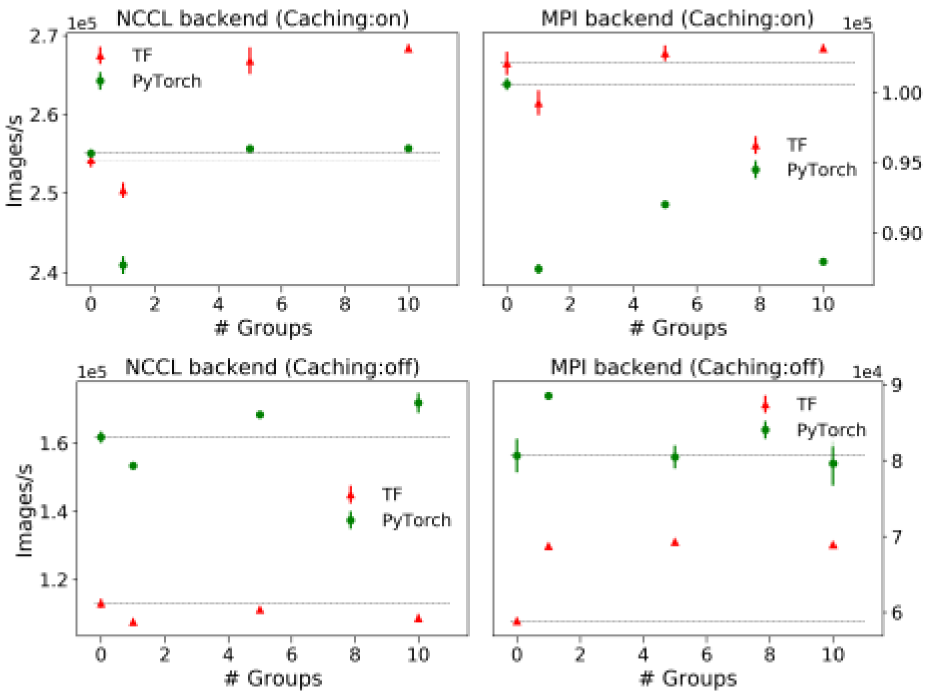


Figure 8: Performance of grouping on ResNet50. We evaluate Horovod with varied grouping configurations on 768 GPUs with caching enabled (top) and disabled (bottom) and with NCCL (left) and MPI (right) backends.

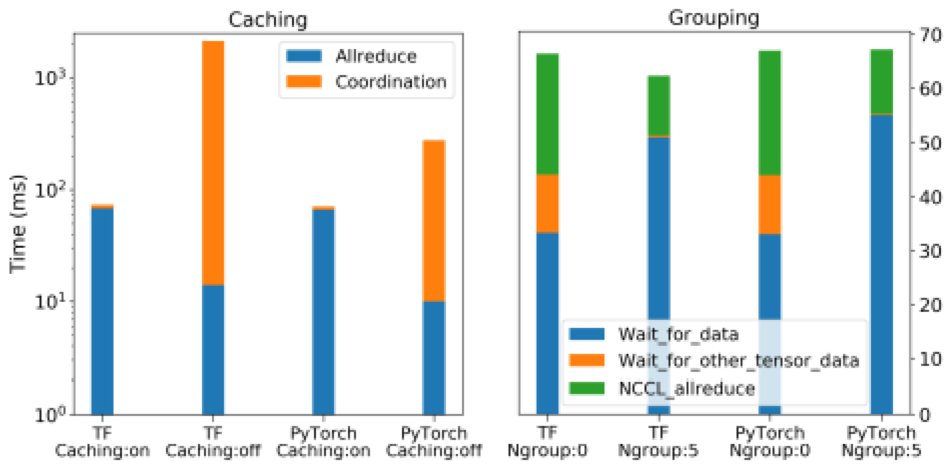


Figure 9: The inner timing breakdown in Horovod (NCCL backend) for a 768-GPU training with caching enabled and disabled (left) and grouping (# groups = 5) (right), respectively, during the training of ResNet50.

优缺点

优点:基于开源的框架进行改进，且文中的实现已经被合进Horovod主分支，且已经发布，说服力较强

缺点:有一些理论分析会更好

相关资料

Horovod: fast and easy distributed deep learning in TensorFlow

源码解析:深度学习分布式训练框架horovod