Blink: A fast NVLink-based collective communication library

Guanhua Wang[†], Amar Phanishayee[⋆], Shivaram Venkataraman[⋆], Ion Stoica[†]

[†]*UC Berkeley* [⋆]*Microsoft Research*

1 INTRODUCTION

Fast training of large deep learning models requires training to scale to many GPUs. Models developed for tasks such as ImageNet 1K can often take days or even weeks to train on a single GPU. The most widely used method for reducing DNN training time is to parallelize the model training process using data-parallel stochastic gradient descent (SGD) [1, 3, 5]. In data-parallel training, each GPU has a full copy of the model parameters and trains independently on a subset of the inputs. GPUs frequently exchange parameters with the other GPUs involved in training.

Synchronizing parameters across GPUs introduces significant overheads when training at scale — a problem accentuated by the fact that GPU computation is getting faster and model sizes are growing larger, thus making communication overheads stand out. Model parameter exchange is often implemented using collective communications primitives such as All-Reduce [2]. The NVIDIA Collective Communications Library (NCCL) [7] is a state-of-the-art implementation that provides inter-GPU collective communication primitives over either PCIe or newer interconnects such as NVLink. By incorporating NCCL into Tensorflow [1], researchers from Uber show that end-to-end training process can be sped up by 60% [9].

However, given a topology, NCCL does not always efficiently use all the available links. This is because NCCL uses ring-based protocols for data transfer and creates as many rings as possible in a given topology. Consider the topology in Figure 1 where we do broadcast from GPUA. Since each link is bi-directional, there are two rings that we can construct as shown in Figure 1(a). Starting from A one ring is A->B->C->D->A, the other in reverse direction is A->C->D->B->A. To do broadcast A can split the data into two parts and send one part on each ring. Thus if the data size is n and link bandwidth size is b, the time taken will be $\frac{n}{2b}$. We note that the two cross-bar links from A<->D and B<->C (shown as dashed lines) are not utilized.

To achieve better link utilization, and consequently faster transfers, we propose `Blink`, a family of protocols that use a broadcast based data transfer scheme and leverage fully connected groups of GPUs to achieve better efficiency. `Blink` also aims to overcome the challenges of topology heterogeneity that can be caused by varying number of GPUs used in training, hardware heterogeneity [4, 6], and multi-machine training. As an example, in Figure 1(b), we show a broadcast protocol in Blink. In this case data sent from GPUA to GPUB is then broadcasted to GPUC and GPUD. We can construct three such forwarding trees and thus the link utilization improves and total time taken becomes $\frac{n}{3b}$.

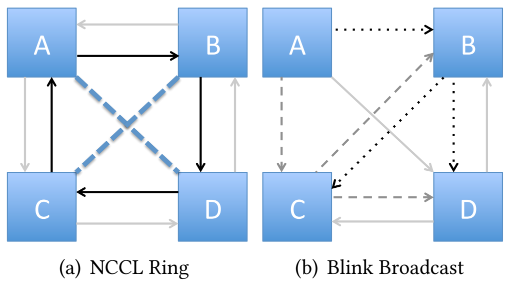
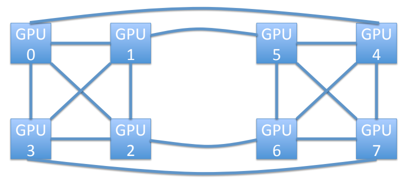


Figure 1: Link usage comparison between Blink and NCCL in fully connected 4 nodes scenario.

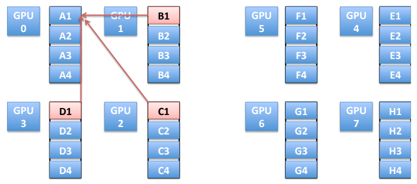
In general, our protocols follow a hierarchical scheme: given a network topology, we first divide the network into groups within which all nodes are fully connected. In the first phase, we perform *internal broadcast* where data is exchanged using all-to-all communication among nodes within each fully connected group. In the second phase we perform *cross-group forwarding*, where we communicate across groups and forward cross group data within the respective groups. We design protocols for four collective communication primitives (Broadcast, Gather, All-Gather, All-Reduce) in `Blink`. Our experiments using up to 8 GPUs on a NVIDIA DGX-1 machine [4] show that, `Blink` can achieve up to 2x speed up compared with state-of-the-art libraries.

2 BACKGROUND

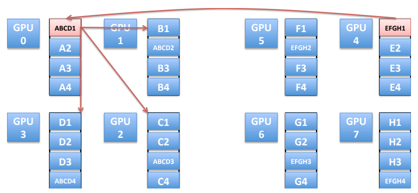
GPU Interconnect Topology: Our main testbed used in this work is the NVIDIA DGX-1, which is an architecture equipped with 8 P100 or V100 GPUs. The GPUs are not only connected by PCIe but also by a newly designed interconnect technology called NVLink [8]. NVLink is a high bandwidth interconnect which can achieve throughput ranging from 20-25 GB/s. As shown in Fig. 2(a), the P100 based DGX-1 has a NVLink topology that consists of a regular hypercube, plus 2 crossbars within each 4-GPU group.



(a) Network Topology of DGX-1's 8 GPUs interconnects with NVLink



(b) Internal broadcast from GPU0



(c) Cross-group forwarding from GPU0

Figure 2: Blink AllReduce protocol on DGX-1

Broadcast vs Ring: Ring-based NCCL data transfer wastes links that cannot form a new ring. Given a network topology, NCCL divides the topology into disjoint rings which go through all the nodes. In most of the circumstances, we have remaining links that cannot form a new ring, leading to idle or wasted links. In our benchmark test of NCCL on a P100 based DGX-1 machine, it can only form one bi-directional ring if we use 4 to 7 GPUs (as shown in Fig. 2(a)), leaving over half of the links idle. For example, if we use 6 GPUs out of 8 GPUs inside the DGX-1 machine, it only uses 6 out of 16 NVLinks.

3 BLINK

We next illustrate how broadcast-based protocols in Blink can improve link utilization and handle variable number of GPUs. For the sake of brevity we limit our discussion to the AllReduce protocol for the DGX-1 topology.

3.1 AllReduce in Blink

Given the DGX-1 topology, our protocol proceeds in a hierarchical fashion. We first divide this topology into two groups, each with 4 fully connected GPUs. We perform a reduction within each group using *internal broadcast* and then communicate across groups with *cross-group forwarding*.

Internal Broadcast: In the internal broadcast phase, we use a reduce-scatter protocol. If we have 4 GPUs in each group, we partition data on each GPU into 4 chunks. We then let GPU1, GPU2, GPU3 transmit their 1st chunk of data (i.e.B1, C1, D1 in Fig. 2(b)) to GPU0, such that after this reduce-scatter step, GPU0 will have the first chunk of the final result within the group (i.e. ABCD1). At the same time, we transmit all the 2nd chunk data to GPU1, so that GPU1 has the final result for the second chunk and so on. At the end of this phase each GPU will have one chunk of the final result from within the group. Note that this scheme applies even when one does not use all the nodes in the group (e.g., when performing all-reduce across 6 GPUs). As a subset of nodes from the fully connected group is also fully connected, we can follow the same internal broadcast described above.

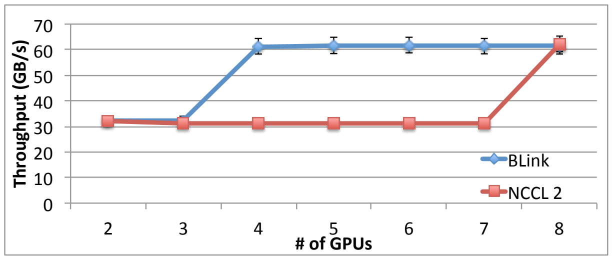


Figure 3: All-Reduce benchmark tests comparison between NCCL 2 and `Blink`.

Cross-group Forwarding: In cross-group forwarding phase, we use cross group links to aggregate the other group's internal result, and forward the final result within the group. In our example, GPU0 receives GPU4's result from the other group (i.e., EFGH1 in Fig. 2(c)), aggregates [1] with its own result (i.e. ABCD1), and then forwards the first chunk of the final result within the group. The whole process is denoted by the line with arrows in Fig. 2(c). Note that this aggregation and forwarding can be performed simultaneously.

3.2 Performance Benchmarks

We next present performance benchmarks for the AllReduce scheme described above. We use a NVIDIA DGX-1 machine and initialize 1GB of data on each GPU. We compare the throughput achieved by NCCL 2 and `Blink` as we vary the number of GPUs from 2 to 8. As shown in Fig. 3, compared with NCCL 2, `Blink` can achieve 2x throughput in 4 to 7 GPUs scenarios. The main reason of this throughput difference is that `Blink` can utilize more NVLinks using our two-step data transfer scheme.

4 FUTURE WORK AND CONCLUSION

This paper proposes `Blink`, a family of broadcast-based collective communication primitives. Compared with ring-based data transfer in NCCL, we show that broadcast based schemes can achieve better link utilization, higher throughput and effectively handle varying number of GPUs. In the future, we plan to extend `Blink` for cross-machine collective communication.

[1]The data aggregation is usually an add function in all-reduce.

REFERENCES

[1] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D. G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., Wicke, M., Yu, Y., and Zheng, X. Tensorflow: A system for large-scale machine learning. In *USENIX OSDI 2016* (2016).

[2] Blaise Barney. Message Passing Interface. https://computing.llnl.gov/tutorials/mpi/.

[3] Dean, J., Corrado, G. S., Monga, R., Chen, K., Devin, M., Le, Q. V., Mao, M. Z., Ranzato, M., Senior, A., Tucker, P., Yang, K., and Ng, A. Y. Large scale distributed deep networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems* (2012), NIPS'12.

[4] NVIDIA DGX-1. https://www.nvidia.com/en-us/data-center/dgx-1, 2017.

[5] Goyal, P., Dollar, P., Girshick, R., Noordhuis, P., Wesolowski, L., Kyrola, A., Tulloch, A., Jia, Y., and He, K. Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour. *arXiv preprint arXiv:1706.02677* (2017).

[6] Nvidia hgx-1 hyperscale gpu accelerator. https://www.nvidia.com/en-us/data-center/hgx-1/.

[7] Jeaugey, S. Optimized inter-GPU collective operations with NCCL 2. https://developer.nvidia.com/nccl, 2017.

[8] NVIDIA NVLINK. http://www.nvidia.com/object/nvlink.html, 2017.

[9] Sergeev, A., and Balso, M. D. Meet Horovod: Uber's Open Source Distributed Deep Learning Framework for TensorFlow. https://eng.uber.com/horovod/, 2017.