BlueConnect: Novel Hierarchical All-Reduce on Multi-tired Network for Deep Learning

Minsik Cho
IBM Research
Austin, TX
minsikcho@us.ibm.com

Ulrich Finkler
IBM Research
Yorktwon Heights, NY
ufinkler@us.ibm.com

David Kung
IBM Research
Yorktwon Heights, NY
kung@us.ibm.com

Abstract

In this paper, we present BlueConnect, an efficient communication library for distributed deep learning that is highly optimized for popular GPU-based platforms. BlueConnect decomposes a single all-reduce operation into a large number of parallelizable reduce-scatter and all-gather operations to exploit the trade-off between latency and bandwidth, and adapt to a variety of network configurations. Therefore, each individual operation can be mapped to a different network fabric and take advantage of the best performing implementation for the corresponding fabric. According to our experimental results, the BlueConnected integrated Caffe2 can significantly reduce synchronization overhead by 87% on 192 GPUs for Resnet-50 training over prior schemes.

1 Introduction

Distributed deep learning is challenging because as the number of learners (or GPUs) increases, the computation time decreases while the amount of communication stays constant Goyal et al. (2017); Uber (2017); You et al. (2017a), resulting in unfavorable computation to communication ratios, and thus diminished returns on more learners. One can either increase the computational workload with a large mini-batch size in stochastic gradient decent (SGD) (i.e., weak scaling) and/or decrease the communication overhead. However, it is known that a large mini-batch beyond a certain point can degrade training quality Balles et al. (2016); Keskar et al. (2016); Krizhevsky (2014), not to mention that mini-batch size is limited by the GPU memory capacity in practice. Therefore, in addition to enabling deep learning with large mini-batch sizes Goyal et al. (2017); Jia et al. (2018); You et al. (2017a,b), it is crucial to develop a fully optimized communication mechanism tuned for deep learning for massive scale-out that can **a)** maximize the bandwidth utilization in popular deep learning environments like GPU-based cluster/cloud, and **b)** minimize the linearly growing communication latency with the number of learners Sridharan et al. (2018).

In this paper, we report the performance of an efficient communication library for deep learning, BlueConnect, that provides a highly efficient all-reduce algorithm for SGD, an integral part in modern deep learning frameworks Abadi et al. (2016); Chen et al. (2015); Facebook (a,b); Goyal et al. (2017); Jia et al. (2014); Niitani et al. (2017); NVidia (2017); Seide & Agarwal (2016). The key idea in BlueConnect is to decompose one all-reduce operation into series of reduce-scatter and all-gather patterns in a topology-aware fashion, which enables a large-scale deep learning with reduced communication overhead. Our technical contribution includes:

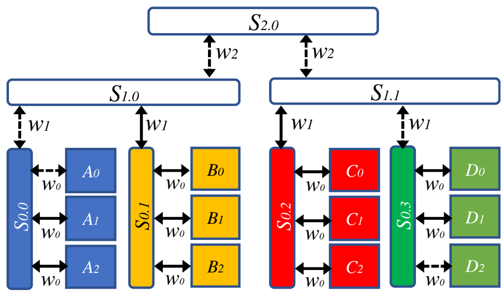
- BlueConnect adapts to the hierarchy of communication bandwidths by leveraging topology-awareness, so that it fully utilizes the heterogeneous network architecture in popular deep learning platforms IBM (2017a); NVidia (a).

32nd Conference on Neural Information Processing Systems (NeurIPS 2018), Montréal, Canada.

- Through topology-aware decomposition, BlueConnect also minimizes the communication latency overhead, the critical bottleneck in large-scale deep learning.
- For each decomposed piece, BlueConnect can mix-and-match various reduce-scatter and all-gather implementations over different network fabrics to maximize network utilization.

2 Preliminaries

2.1 Notations and Basic Performance Models



α	non-zero latency time per transfer at each network switch
w_i	bandwidth of network switch type i
$s_{i.j}$	a network switch instance j with w_i
W	a set of network bandwidths $\{w_i \mid \exists i \in \mathbb{Z}_{\geq 0}\}$
S	a set of switch instances $\{s_{i.j} \mid w_i \in W, \exists j \in \mathbb{Z}_{\geq 0}\}$
P	a set of learners
N	the gradient's size in unit

Figure 1: 4 nodes with 12 learners on heterogeneous network architecture connected hierarchically and their notations.

Notations used in this paper are listed in Fig 1. For a given data size n, a learner count p, and a bandwidth w, the performance of a ring-based or recursive halving/doubling communication pattern can be expressed as follows Thakur et al. (2005):

$$\mathcal{T}_{r/c}(p, n, \alpha, w) = \begin{cases} \mathcal{T}_c(p, n, \alpha, w) = \lg(p)\alpha + \frac{p-1}{p}\frac{n}{w} & p = 2^q, q \in \mathbb{Z} \\ \mathcal{T}_r(p, n, \alpha, w) = (p-1)\alpha + \frac{p-1}{p}\frac{n}{w} & \text{otherwise /*ring*/} \end{cases} \quad (1)$$

Based on the ring and recursive communication patterns, we can compute the communication performance of `broadcast` or `reduce` as follows Thakur et al. (2005):

$$\begin{aligned} \mathcal{T}_{bcast}(p, n, \alpha, w) &= \mathcal{T}_{reduce}(p, n, \alpha, w) \\ &= \mathcal{T}_c(p, n, \alpha, w) + \mathcal{T}_r(p, n, \alpha, w) \end{aligned} \quad (2)$$

Since our focus is on heterogeneous network architecture Dichev & Lastovetsky (2014), we extend the homogeneous model Thakur et al. (2005) by using different w_i. For example, we assume a typical hierarchically built cluster over tree-like heterogeneous network architecture NVidia (b) as in Fig. 1 where 12 learners ($P = \{A_i, B_i, C_i, D_i \mid \forall i \in \{0, 1, 2\}\}$) are connected through heterogeneous network switches in $S = \{s_{0.\{0,1,2,3\}}, s_{1.\{0,1\}}, s_{2.0}\}$. Regarding the example in Fig 1, $s_{0.*}$ can represent an intra-node network like NVLink around 32GB/s per lane, while $s_{1.*}$ and $s_{2.0}$ may represent inter-node switches for 100Gbps InfiniBand. In such cases, w_1 and w_2 would be 100Gbps and 200Gbps respectively to ideally match the total uplink bandwidth from all the hanging nodes (e.g., fat-tree Al-Fares et al. (2008); NVidia (b)).

2.2 All-Reduce for Distributed SGD

The key communication pattern used in SGD synchronization in deep learning is `all-reduce` Amodei et al. (2015); Baidu (2017) which is popularly implemented with ring-based `reduce_scatter` or `all_gather` Thakur et al. (2005). Based on Eq.(1, 2), the synchronization costs of prior arts in deep learning can be computed. For example, one-level ring-based `all-reduce` in Baidu (2017); Thakur et al. (2005) can be expressed with the following performance model:

$$T_{one\_lvl} = 2(|P| - 1)\{\alpha + \frac{\frac{N}{|P|}}{\min_{0 \leq i < |W|}\{w_i\}}\} = 2\mathcal{T}_r(|P|, N, \alpha, \min W) \quad (3)$$

where there are $2(|P|-1)$ iterations in Eq. (3), and each iteration needs to transfer $\frac{N}{|P|}$ data over $w_0, w_1, ..., w_{|W|-1}$ in the worst case (e.g., marked with dotted arrows from A_0 to D_2 in Fig. 1). Although one-level ring-based `all-reduce` has been widely used for traditional high-performance computing, it is not quite suitable for large-scale deep learning for two reasons:

- A node with multiple GPUs (up to 16 GPUs per node Amazon) may have multiple learners inside and increases $|P|$ fast, which would rapidly increase the latency of deep learning communication (i.e., a large multiplier to α).
- Since deep learning typically runs on a heterogeneous network topology (e.g., Fig. 1), the performance of one-level approach is gated by the slowest bandwidth along the path (i.e., $\min W$), not fully utilizing other fast networks fabrics.

To address this problem, a two-level approach is used in the state-of-the-art deep learning softwares Facebook (b); Jia et al. (2018); NVidia (2017). In the first step, the gradients are reduced to the master learner on each node. Then, a *small-scale* one-level ring-based `all-reduce` is applied among the master learners only. Finally, the gradient in the master learners is locally broadcast back to the other learners within the same node, synchronizing all the learners in the training task. When $|P|$ is decomposed into two learner counts such as p_0 (the number of learners within each node) and p_1 (the number of master learners) like $|P| = p_0 p_1$, the performance of such a two-level scheme can be formally expressed as follows Jia et al. (2018):

$$\begin{aligned} T_{two\_lvl} &= \mathcal{T}_{reduce}(p_0, N, \alpha, w_0) + \mathcal{T}_{bcast}(p_0, N, \alpha, w_0) + 2\mathcal{T}_{r/c}(p_1, N, \alpha, \min_{0 \le i < |W|}\{w_i\}) \\ &= 2\mathcal{T}_c(p_0, N, \alpha, w_0) + 2\mathcal{T}_r(p_0, N, \alpha, w_0) + 2\mathcal{T}_{r/c}(\frac{P}{p_0}, N, \alpha, \min W) \end{aligned} \quad (4)$$

We can trivially show that Eq. (4) has smaller latency overhead than Eq. (3), yet it would still suffer from the following three limitations: **a)** latency overhead can be large when $p_0 \ll |P|$, **b)** performance is still gated by $\min W$, **c)** many learners stay idle during the 2nd step, leading to bandwidth under-utilization. Our proposed BlueConnect in Section 3 addresses these limitations with a novel topology-aware scheme.

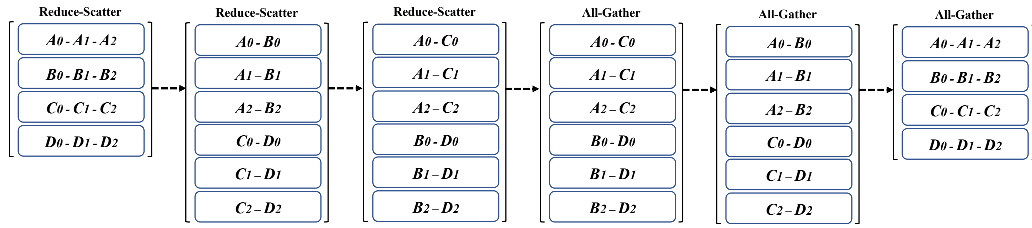
3 BlueConnect

BlueConnect decomposes `all-reduce` to fit into heterogeneous network hierarchy and increase the hardware utilization. One well-known way of decomposing `all-reduce` is to use `reduce-scatter` followed by `all-gather` which are popularly implemented based on the ring scheme. Such crude decomposition has neither granularity nor flexibility sufficient enough to utilize the underlying hardware and the highly optimized implementations (i.e., ones offered by the hardware vendors) efficiently. We, however, found that the `reduce-scatter` and `all-gather` can be further decomposed into multiple stages of parallelizable `reduce-scatter` and `all-gather` operations if the following integer factorization exists:

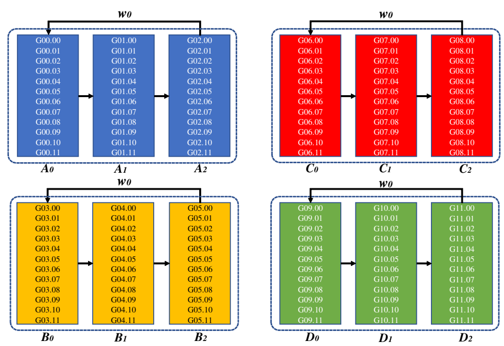
$$|P| = p_0 p_1 p_2 ... p_k = \prod_{i<k} p_i \quad (p_i \in N, p_i > 1) \quad (5)$$

In detail, the `reduce-scatter` can be further decomposed into the k stages of bundled `reduce-scatter` operations where the i-th stage has $\frac{P}{p_i}$ *concurrently* launchable `reduce-scatter` operations over different subsets of learners. The `all-gather` can also be further decomposed in the same way, but they have a backward dependency. If `all-reduce` is performed based on the proposed decomposition, every learner participates in one of the `reduce-scatter` or `all-gather` operations at any moment or stage (unlike the two-step approach). The strength of the proposed decomposition are two-fold: **a)** decomposition can offer enough granularity and flexibility to map operations to underlying network elements and implementations, **b)** higher parallelism at each stage can increase the bandwidth utilization Sivakumar et al. (2000); Yildirim et al. (2016).

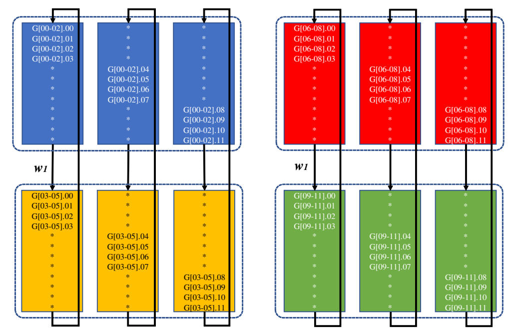
While `all-reduce` can be decomposed into various ways, BlueConnect does so to optimize against the network topology. First, BlueConnect decomposes `all-reduce` into the same number of `reduce-scatter` and `all-gather` stages as the number of network hierarchy levels (i.e., k



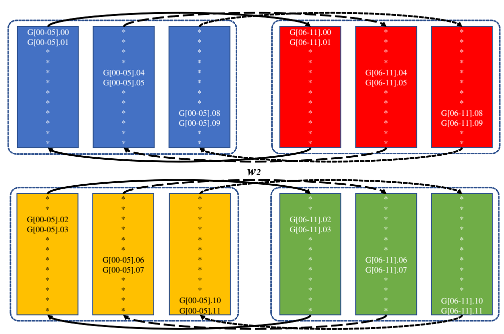
(a) `all-reduce` is decomposed into the multiple stages of `reduce-scatter` /`all-gather` operations.



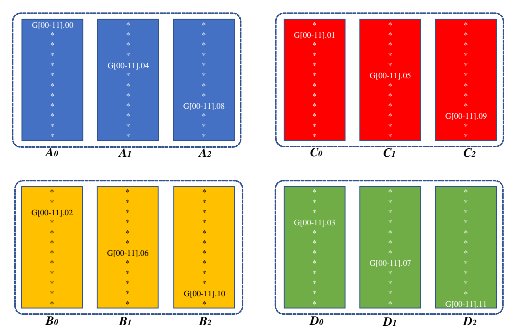
(b) 4 parallel `reduce-scatter` operations with w_0



(c) 6 parallel `reduce-scatter` operations with w_1



(d) 6 parallel `reduce-scatter` operations with w_2



(e) the final `reduce-scatter` result

Figure 2: BlueConnect `reduce-scatter` example for 12 GPUs with $|P| = p_0p_1p_2$ where $p_0 = 3, p_1 = 2$, and $p_2 = 2$. The reverse steps with `all-gather` shall be taken to complete `all-reduce`.

in Eq. (5)). Then, the amount of parallelism in each stage is determined by the number of elements in each network hierarchy level. Fig. 2 (a) shows an example where BlueConnect decomposes $|P| = p_0p_1p_2 = 3 \times 2 \times 2$ mapping to w_0, w_1, and w_2 respectively for Fig. 1, because there are 3 GPUs within a node, forming a binary tree. This way, BlueConnect can avoid the bandwidth bottleneck in the ring-based scheme (see Eq. (3)).

Once decomposition is completed, BlueConnect executes `reduce-scatter` and `all-gather` operations on various partitions of the input data, in a MPI-compliant manner. Considering `all-reduce` for Fig. 1, BlueConnect performs the following steps:

Fig. 2 (b): Four `reduce-scatter` operations are performed concurrently with w_0 and within a node. Note that data size for each instance is N.

Fig. 2 (c): Six short `reduce-scatter` operations are performed concurrently with w_1. $A_{\{0,1,2\}} \rightarrow B_{\{0,1,2\}}$ run over $s_{1.0}$, while $C_{\{0,1,2\}} \rightarrow D_{\{0,1,2\}}$ run over $s_{1.1}$, all concurrently. Note that data size for each instance is $\frac{N}{3}$.

Fig. 2 (d): Six `reduce-scatter` operations, $A_{\{0,1,2\}} \rightarrow C_{\{0,1,2\}}$ and $B_{\{0,1,2\}} \rightarrow D_{\{0,1,2\}}$ are performed concurrently over $s_{2.0}$ with w_2, yet the data size for each instance is only $\frac{N}{6}$.

Fig. 2 (e): All the `reduce-scatter` stages are completed, and the reduced gradients are evenly distributed. `all-gather` will begin in the exactly same but reverse order to complete `all-reduce`.

As in Fig. 2, BlueConnect fully leverages the heterogeneous network bandwidths with inexpensive multiple/concurrent `reduce-scatter` and `all-gather` operations. BlueConnect distributes data over all available nodes, which provides the following key differences from the two-level scheme:

- BlueConnect decomposes P according to the network topology and hierarchy. The goal of such decomposition is to keep traffic within each switch level as much as possible, in order to reduce the hop count and maximize the bandwidth utilization on each switch.
- BlueConnect reduces the latency overhead through decomposition. Such decomposition in BlueConnect also enables to use recursive halving/double approaches for non-power-of-two $|P|$. For example, if $|P| = 96$, the two-level approaches cannot use recursive halving/double (without expensive preprocessing), but BlueConnect can decompose into $|P| = 16 \times 6$ and use recursive methods for the first `reduce-scatter` and the last `all-gather` stages to further reduce latencies.
- BlueConnect runs multiple ring communication patterns over a single link, maximizing bandwidth utilization Sivakumar et al. (2000); Yildirim et al. (2016). $A_{\{0,1,2\}} \rightarrow B_{\{0,1,2\}}$ run over w_1 concurrently in Fig. 2 (a). Such multiple parallel rings easily sustain full link utilization, leaving no idle time. We found BlueConnect hit the near-theoretical bandwidth limit in most cases, while a single ring does not.
- The multiple ring patterns in BlueConnect obviously require learners to share switches. In Fig. 2 (a), six disjoint sets of ring communication patterns, $A_{\{0,1,2\}} \rightarrow C_{\{0,1,2\}}$ and $B_{\{0,1,2\}} \rightarrow D_{\{0,1,2\}}$ share $s_{2.0}$, leaving $\frac{w_2}{6}$ to each stream. Such reduced bandwidth per stream is compensated by the reduced amount of data to transfer (i.e., $\frac{N}{6}$). Since BlueConnect exercises all learners at any moment and each learner sends data to a single leaner, we can easily compute the bandwidth fraction for each ring by dividing the bandwidth by the number of learners under the corresponding network hierarchy (e.g., $\frac{w_1}{3}$ and $\frac{w_2}{6}$).

Assume topology-aware decomposition $P = \prod_{j=0}^{|W|-1} p_j$ for a fat-tree like topology as in Fig. 1. The performance model of BlueConnect can be stated as in Eq. (6). We can easily prove that BlueConnect offers smaller latency than the two-level scheme in Eq. (4).

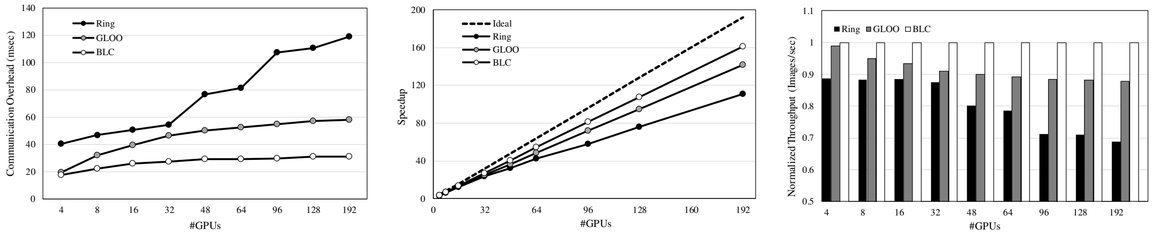
$$T_{blc} = 2\mathcal{T}_{r/c}(p_0, N, \alpha, w_0) + 2\mathcal{T}_{r/c}(p_1, \frac{N}{p_0}, \alpha, \min\{w_0, \frac{w_1}{p_0}\}) + ... \tag{6}$$

$$= 2\sum_{i=0}^{|W|-1} \mathcal{T}_{r/c}(p_i, \frac{N}{\prod_{j=0}^{i-1} p_j}, \alpha, \min_{0 \le j < i}\{\frac{w_j}{\prod_{k=0}^{j-1} p_k}\}) \tag{7}$$

4 Experimental Results

We implemented BlueConnect (**BLC**)[1] for GPU in C++ based on CUDA-aware MPI IBM (2017b) and NCCL ver. 2 NVidia (2017) (without using `all-reduce` APIs) to exchange gradients efficiently. BLC picks the best performing `reduce-scatter` and `all-gather` implementation directly from MPI and NCCL, or from custom implementations for each network fabric. Our **BLC** implementation is packaged as a new communication operator for Caffe2 Facebook (a), following existing communication operator implementation. To evaluate the performance of **BLC**, we used a cluster of 48 IBM S822LC systems on Red Hat Enterprise Linux with cuDNN, each equipped with 4 NVidia Tesla P100-SXM2 GPUs connected through NVLink IBM (2017a); NVLink (2017). The systems were organized into 3 racks with 16 nodes each, connected via a single-port 100Gbs InfiniBand network. We compared **BLC** with the following deep learning communication techniques/libraries under the identical environment:

[1] the latest version is available through IBM PowerAI DDL



(a) Communication Overhead (b) Scaling Efficiency (c) Training Throughput

Figure 3: Training performance comparison over 192 GPUs

MPI_Allreduce `all-reduce` function in MPI Thakur et al. (2005)

Ring One-level ring-based `all-reduce` algorithm as in Baidu (2017), designed for deep learning.

GLOO Two-level `all-reduce` algorithm in Facebook (b); Jia et al. (2018), designed for deep learning based on NCCL NVidia (2017) and ib_verb.

We used Resnet-50 Goyal et al. (2017); He et al. (2015) and ImageNet-1K to measure scaling efficiency and communication overheads for 4 GPUs, 8 GPUs, up to 192 GPUs, while maintaining a fixed batch size of 32 per GPU (e.g., the effective batch size is 6144 at 192 GPUs). We found that Resnet-50 has about 100MB of gradients in FP32. Since Goyal et al. (2017); You et al. (2017a) has demonstrated successful convergence to best accuracy for the batch size of 8192, the scaling efficiency number is meaningful. We confirmed that our **BLC** integration into Caffe2 does not alter the convergence behavior through several tests.

We present our results in Fig. 3 without **MPI_Allreduce** results (due to its poor performance beyond 32 GPUs). To accurately measure the communication overhead (actual `all_reduce` time, interface-overhead to Caffe2, jitter from network/OS/GPU-scheduling, required memory copy, and so on), we first measure the single-GPU performance which is 163.0 *msec* per iteration or 196.3 images/sec. Our experimental results in Fig. 3 are summarized as follows:

- (a) plots the overall communication overhead per iteration over various GPU counts. We subtracted the baseline number (163.0 *msec* as mentioned above) to capture the total communication overhead reliably and comprehensively. With 4 GPUs (which are all in a single node), **BLC** and **GLOO** show similar performance because both simply use NCCL (while **Ring** does not). However, **BLC** incurs less communication overhead with more GPUs.
- The communication overhead in (a) for **BLC** on 192 GPUs is about 31.0 *msec* where the jitter accounts for 5-10 *msec*. **GLOO** scales much better than **Ring**, but **BLC** offers the best scaling overall, with about 87% reduction in communication overhead over **GLOO** on 192 GPUs (58.0 vs 31.0 *msec*). If we assume the jitter is 5 *msec*, then the actual communication overhead improvement of **BLC** over **GLOO** is about $2\times$ on 192 GPUs.
- (b) highlights how communication overhead impacts the scaling efficiency, one of the key metrics in large-scale deep learning. Note that our scaling efficiency is compared with respect to a single-GPU performance, instead of a single-node performance Goyal et al. (2017). It shows that **BLC** scales best due to efficient synchronization in SGD. **Ring** scales worst, keeping GPUs idle for an extended period; it wastes 48% of GPU computing power on 192 GPUs (equivalent to 92 GPUs).
- (c) shows that **BLC** delivers the best images/sec throughput over other communication techniques. On 192 GPUs, **BLC** delivers 11% higher throughput than **GLOO**, and 45% higher throughput than **Ring**.

5 Conclusion and Future Work

We have proposed BlueConnect, an efficient communication library for training complex deep neural networks with a large number of GPUs, thus offering a viable strategy to reduce training time from weeks to hours. Such rapid turn around can accelerate the improvement of existing neural networks and design of new neural networks, and exploration of new application domains.

6 Acknowledgment

We thank IBM Power AI team for assistance with BlueConnect implementation and testing, and Brad Neimanich, Alex Habeger, Bryant Nelson, Nicolas Castet, Bill Armstrong for BlueConnect integration and productization into IBM PowerAI DDL.

References

Abadi, Martin, Barham, Paul, Chen, Jianmin, Chen, Zhifeng, Davis, Andy, Dean, Jeffrey, Devin, Matthieu, Ghemawat, Sanjay, Irving, Geoffrey, Isard, Michael, Kudlur, Manjunath, Levenberg, Josh, Monga, Rajat, Moore, Sherry, Murray, Derek G., Steiner, Benoit, Tucker, Paul, Vasudevan, Vijay, Warden, Pete, Wicke, Martin, Yu, Yuan, and Zheng, Xiaoqiang. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pp. 265–283, 2016.

Al-Fares, Mohammad, Loukissas, Alexander, and Vahdat, Amin. A scalable, commodity data center network architecture. *SIGCOMM Comput. Commun. Rev.*, 38(4):63–74, August 2008.

Amazon. https://aws.amazon.com/ec2/instance-types/p2.

Amodei, Dario, Anubhai, Rishita, Battenberg, Eric, Case, Carl, Casper, Jared, Catanzaro, Bryan, Chen, Jingdong, Chrzanowski, Mike, Coates, Adam, Diamos, Greg, Elsen, Erich, Engel, Jesse, Fan, Linxi, Fougner, Christopher, Han, Tony, Hannun, Awni Y., Jun, Billy, LeGresley, Patrick, Lin, Libby, Narang, Sharan, Ng, Andrew Y., Ozair, Sherjil, Prenger, Ryan, Raiman, Jonathan, Satheesh, Sanjeev, Seetapun, David, Sengupta, Shubho, Wang, Yi, Wang, Zhiqian, Wang, Chong, Xiao, Bo, Yogatama, Dani, Zhan, Jun, and Zhu, Zhenyao. Deep speech 2: End-to-end speech recognition in english and mandarin. *CoRR*, abs/1512.02595, 2015.

Baidu. https://github.com/baidu-research/baidu-allreduce. 2017.

Balles, Lukas, Romero, Javier, and Hennig, Philipp. Coupling adaptive batch sizes with learning rates. *CoRR*, abs/1612.05086, 2016.

Chen, Tianqi, Li, Mu, Li, Yutian, Lin, Min, Wang, Naiyan, Wang, Minjie, Xiao, Tianjun, Xu, Bing, Zhang, Chiyuan, and Zhang, Zheng. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *CoRR*, abs/1512.01274, 2015.

Dichev, K. and Lastovetsky, A. *Optimization of collective communication for heterogeneous HPC platforms*, pp. 95–114. Wiley Series on Parallel and Distributed Computing. 2014.

Facebook. https://caffe2.ai. a.

Facebook. https://github.com/facebookincubator/gloo. b.

Goyal, Priya, Dollár, Piotr, Girshick, Ross B., Noordhuis, Pieter, Wesolowski, Lukasz, Kyrola, Aapo, Tulloch, Andrew, Jia, Yangqing, and He, Kaiming. Accurate, large minibatch SGD: training imagenet in 1 hour. *CoRR*, abs/1706.02677, 2017.

He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

IBM. https://www.ibm.com/us-en/marketplace/high-performance-computing. 2017a.

IBM. https://www.ibm.com/us-en/marketplace/spectrum-mpi. 2017b.

Jia, Xianyan, Song, Shutao, He, Wei, Wang, Yangzihao, Rong, Haidong, Zhou, Feihu, Xie, Liqiang, Guo, Zhenyu, Yang, Yuanzhou, Yu, Liwei, Chen, Tiegang, Hu, Guangxiao, Shi, Shaohuai, and Chu, Xiaowen. Highly scalable deep learning training system with mixed-precision: Training imagenet in four minutes. 2018.

Jia, Yangqing, Shelhamer, Evan, Donahue, Jeff, Karayev, Sergey, Long, Jonathan, Girshick, Ross, Guadarrama, Sergio, and Darrell, Trevor. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

Keskar, Nitish Shirish, Mudigere, Dheevatsa, Nocedal, Jorge, Smelyanskiy, Mikhail, and Tang, Ping Tak Peter. On large-batch training for deep learning: Generalization gap and sharp minima. *CoRR*, abs/1609.04836, 2016.

Krizhevsky, Alex. One weird trick for parallelizing convolutional neural networks. *CoRR*, abs/1404.5997, 2014.

Niitani, Yusuke, Ogawa, Toru, Saito, Shunta, and Saito, Masaki. Chainercv: a library for deep learning in computer vision. *CoRR*, abs/1708.08169, 2017.

NVidia. https://devblogs.nvidia.com/parallelforall/dgx-1-fastest-deep-learning-system. a.

NVidia. https://www.nvidia.com/en-us/data-center/dgx-saturnv. b.

NVidia. https://developer.nvidia.com/nccl. 2017.

NVLink. https://en.wikipedia.org/wiki/NVLink. 2017.

Seide, Frank and Agarwal, Amit. Cntk: Microsoft's open-source deep-learning toolkit. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pp. 2135–2135, 2016.

Sivakumar, H., Bailey, S., and Grossman, R. L. Psockets: The case for application-level network striping for data intensive applications using high speed wide area networks. In *Proceedings of the 2000 ACM/IEEE Conference on Supercomputing*, 2000.

Sridharan, Srinivas, Vaidyanathan, Karthikeyan, Kalamkar, Dhiraj, Das, Dipankar, Smorkalov, Mikhail E., Shiryaev, Mikhail, Mudigere, Dheevatsa, Mellempudi, Naveen, Avancha, Sasikanth, Kaul, Bharat, and Dubey, Pradeep. On scale-out deep learning training for cloud and hpc. 2018.

Thakur, Rajeev, Rabenseifner, Rolf, and Gropp, William. Optimization of collective communication operations in mpich. *Int. J. High Perform. Comput. Appl.*, 19(1):49–66, February 2005.

Uber. https://eng.uber.com/horovod. 2017.

Yildirim, E., Arslan, E., Kim, J., and Kosar, T. Application-level optimization of big data transfers through pipelining, parallelism and concurrency. *IEEE Transactions on Cloud Computing*, 4(1): 63–75, 2016.

You, Yang, Gitman, Igor, and Ginsburg, Boris. Scaling SGD batch size to 32k for imagenet training. *CoRR*, abs/1708.03888, 2017a.

You, Yang, Zhang, Zhao, Hsieh, Cho-Jui, Demmel, James, and Keutzer, Kurt. Imagenet training in minutes. 2017b.