BlueConnect: Decomposing All-Reduce for Deep Learning on Heterogeneous Network Hierarchy

Minsik Cho [1] **Ulrich Finkler** [2] **David Kung** [2] **Hillery Hunter** [2]

Abstract

As deep neural networks get more complex and input datasets get larger, it can take days or even weeks to train a deep neural network to the desired accuracy. Therefore, enabling distributed deep learning at a massive scale is critical, since it offers the potential to reduce the training time from weeks to hours. 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 on two system configurations, BlueConnect can outperform the leading industrial communication library by wide margin, and the BlueConnect integrated Caffe2 can significantly reduce synchronization overhead by 87% on 192 GPUs for Resnet-50 training over prior schemes.

1 Introduction

Deep learning has become the de-facto technique for an increasing number of cognitive applications, including vision, speech, and language translation (Amodei et al., 2015; Ioffe & Szegedy, 2015; Jia et al., 2014). The success is driven by the availability of an enormous volume of data and advances in deep neural networks, which in turn make deep learning one of the most computationally demanding AI applications (Amodei et al., 2015; Chen et al., 2016; Krizhevsky et al., 2012). Hardware accelerators such as GPU/TPU and their accompanying software stacks have provided a significant amount of speed up (Jouppi et al., 2017; NVidia, 2017b). However, deep neural network training for speech and vision can still take days and even weeks. Therefore, parallelization by distributing the deep learning training to many (upwards of hundreds) GPUs over a cluster or on a cloud environment is critical to cut the training time from weeks to hours and minutes (Goyal et al., 2017; Iandola et al., 2015; Jia et al., 2018; You et al., 2017a;b).

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, 2017a; 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 com-

[1]IBM Systems, Austin, Texas, USA [2]IBM T. J. Watson Research Center, Yorktown Heights, New York, USA. Correspondence to: Minsik Cho <minsikcho@us.ibm.com>.

Proceedings of the 2^{nd} SysML Conference, Palo Alto, CA, USA, 2019. Copyright 2019 by the author(s).

munication 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).

- 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/algorithms over different network fabrics to maximize network utilization.

The rest of the paper is organized as follows. We present preliminaries in Section 2. Section 3 discusses our proposed algorithm, BlueConnect. Experimental results are in Section 4, followed by the conclusion in Section 5.

2 PRELIMINARIES

2.1 Prior Arts

To enable large scale distributed deep learning with hundreds of GPUs under popular data-parallelism (Amodei et al., 2015; Goyal et al., 2017; You et al., 2017a), the batch size must be in the thousands since GPU utilization and compute to communication ratio are low for single digit batch size per GPU for typical neural networks. Since a large batch size in deep learning may cause poor convergence, there have been recent efforts to mitigate such convergence and generalization issues (Goyal et al., 2017; Keskar et al., 2016; You et al., 2017a). (Goyal et al., 2017) proposed a linear learning rate scaling rule and performed learning rate warm-up from a small/safe value to the larger target value in the early training phase, and then resorting to the usual step-wise descent. For gradient synchronization, (Goyal et al., 2017) leveraged a deep learning communication library (Facebook, b) to demonstrate that Resnet-50 (He et al., 2015) can be trained in one hour over 32 DGX-1's (256 GPUs). Recently, (Jia et al., 2018) demonstrated that the same Resnet50 can be trained in four minutes over 1024 GPUs with low-precision (i.e., FP16).

While this is an impressive result and there exist numerous communication algorithms for distributed computing platforms (Almási et al., 2005; Baidu, 2017; Jia et al., 2018; Thakur et al., 2005), they are not necessarily customized and optimized for large scale distributed deep learning (Amodei et al., 2015): **a)** most existing techniques were developed for a homogeneous environment, while deep learning will be increasingly deployed on a heterogeneous environment like cloud, **b)** the traffic generated by deep learning is highly bursty and extremely large (i.e., 100MB -1GB), while existing techniques have been optimized for relatively small and frequent exchanges, and **c)** most existing algorithms are not tuned for new network fabrics (i.e, NVLink (NVLink, 2017)). As future GPUs/accelerators double their performance each generation, the gradient synchronization in SGD will become a considerable bottleneck in large-scale deep learning (Keuper, 2016). Hence, it is in great demand to study an efficient communication technique for deep learning that addresses the 3 issues mentioned above.

Table 1. Notations.

α	non-zero latency time per transfer at each network switch
w_i	bandwidth (unit/sec) of network switch type i
$s_{i.j}$	a network switch instance j with w_i
W	a set of network bandwidths, $\{w_i \| \exists i \in \mathbb{Z}_{\geq 0}\}$
S	a set of switch instances, $\{s_{i.j} \| w_i \in W, \exists j \in \mathbb{Z}_{\geq 0}\}$
P	a set of learners
N	the gradient's size in unit
$c(r, w)$	a set of learners that perform reduce_scatter and all_gather over bandwidth w with a learner r

Other approaches to reduce communication overhead in deep learning are largely based on approximation of fully synchronous SGD (Wang & Joshi, 2018) including asynchronous SGD (ASGD) where each learner can subscribe the updated weights from a parameter server asynchronously (i.e., removing the synchronization barrier and suppressing bursty traffic) (Niu et al., 2011; Zhang et al., 2015a;b) and decentralized SGD where each learner communicates only with a subset of all learners (Lian et al., 2017). It is shown that such approximated or stochastic methods can improve the scalability of distributed deep learning but at a cost of potentially reduced accuracy and instable convergence (Chen et al., 2016).

2.2 Notations and Basic Performance Models

Notations used in this paper are listed in Table 1. We ignore arithmetic operation time, as it is trivially cheap in deep learning on GPUs. Then, for a given data size n, a learner count p, and a bandwidth w, the performance of a ring-based communication pattern can be expressed as follows (Thakur et al., 2005):

$$\mathcal{T}_r(p, n, \alpha, w) = (p-1)\alpha + \frac{p-1}{p}\frac{n}{w} \tag{1}$$

When the latency between any two nodes is uniform and p is a power-of-two number, one can use recursive halving/doubling to obtain the same result with smaller latency, which can expressed as follows (Thakur et al., 2005):

$$\mathcal{T}_c(p, n, \alpha, w) = \lg(p)\alpha + \frac{p-1}{p}\frac{n}{w} \tag{2}$$

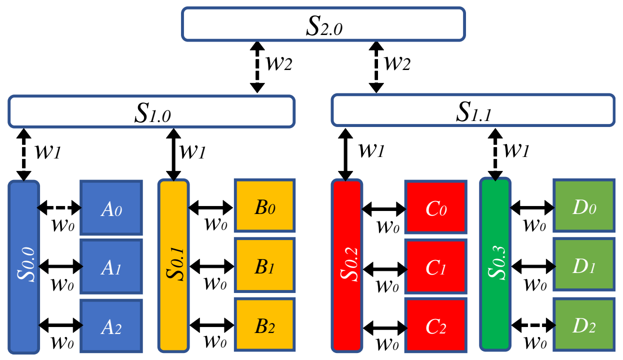


Figure 1. 4 nodes with 12 learners on heterogeneous network architecture connected hierarchically.

By combining Eq. (1) and (2), we define the following to get the best of both:

$$\mathcal{T}_{r/c}(p,n,\alpha,w) = \begin{cases} \mathcal{T}_c(p,n,\alpha,w) & p = 2^q, q \in \mathbb{Z} \\ \mathcal{T}_r(p,n,\alpha,w) & \text{otherwise} \end{cases} \quad (3)$$

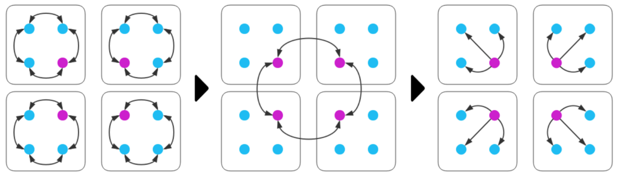
Based on the ring and recursive communication patterns, we can compute the communication performance of `broadcast` or `reduce` as follows (Thakur et al., 2005):

$$\mathcal{T}_{bcast}(p,n,\alpha,w) = \mathcal{T}_{reduce}(p,n,\alpha,w)$$
$$= \mathcal{T}_c(p,n,\alpha,w) + \mathcal{T}_r(p,n,\alpha,w) \quad (4)$$

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 ($P = \{A_i, B_i, C_i, D_i | \forall i \in \{0,1,2\}\}$) 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)).

2.3 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.(1,4), the synchronization costs of prior arts in deep learning can be computed. For example of one-level ring-based `all-reduce` in (Baidu, 2017; Thakur et al., 2005) can be expressed with the following performance model:



Intra Ring over PCIE/NVLink
Inter Ring over GPU Direct RDMA
Broadcast over PCIE/NVLink

Figure 2. Two-level `all-reduce` (Jia et al., 2018) where only master learners are active in the 2nd step.

$$T_{one\_lvl} = 2(|P|-1)\{\alpha + \frac{\frac{N}{|P|}}{\min_{0 \le i < |W|}\{w_i\}}\} \quad (5)$$
$$= 2\mathcal{T}_r(|P|, N, \alpha, \min W) \quad (6)$$

where there are $2(|P|-1)$ iterations in Eq. (5), 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, 2017a) shown in Fig. 2. 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):

$$\begin{aligned} T_{two\_lvl} &= \mathcal{T}_{reduce}(p_0, N, \alpha, w_0) \text{*reduce to master*} \\ &\quad + \mathcal{T}_{bcast}(p_0, N, \alpha, w_0) \quad \text{*bcast from master*} \\ &\quad + 2\mathcal{T}_{r/c}(p_1, N, \alpha, \min_{0\le i<|W|}\{w_i\}) \\ &= 2\mathcal{T}_c(p_0, N, \alpha, w_0) + 2\mathcal{T}_r(p_0, N, \alpha, w_0) \\ &\quad + 2\mathcal{T}_{r/c}(\frac{P}{p_0}, N, \alpha, \min W) \end{aligned} \tag{7}$$

Although we can trivially show that Eq. (7) has smaller latency overhead than Eq. (6), it would still suffer from the following three limitations:

- Latency overhead can be large when $p_0 \ll |P|$.
- Performance is still gated by $\min W$.
- Many learners stay idle during the 2nd step, leading to bandwidth under-utilization.
- `reduce`/`broadcast` at the first step and the last step is expensive.

Our proposed BlueConnect in Section 3 addresses these limitations with a novel topology-aware scheme as in Section 3.2 based on the `all-reduce` decomposition in Section 3.1.

3 BlueConnect

In this section, we introduce a communication library for deep learning, BlueConnect, with detailed examples. The main goal of BlueConnect is to greatly reduce the communication/synchronization overhead for massive scale-out of deep learning based on topology-aware `all-reduce`. In contrast to the prior arts in Section 2, BlueConnect relies on a series of multiple and concurrent `reduce-scatter` and `all-gather` operations and generates traffic patterns optimized for heterogeneous network topology, leveraging full network capacity. We assume tree-topology for network architecture for illustration purpose, but the BlueConnect is flexible enough to be mapped to other architectures including mesh/torus network (Almási et al., 2005) as well (see Section 3.3). Section 3.1 focuses on `all-reduce` decomposition, and Section 3.2 formally presents BlueConnect with the performance model given in Section 3.3.

3.1 All-Reduce Decomposition

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

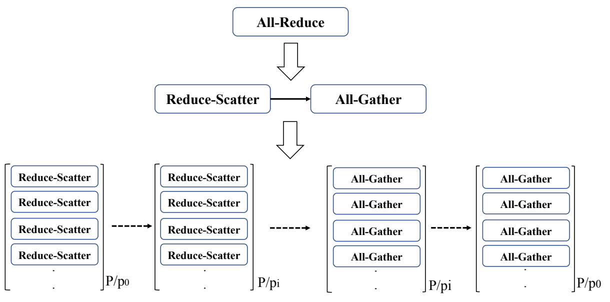


Figure 3. All-reduce can be decomposed into multiple stages of parallelizable reduce-scatter and all-gather operations.

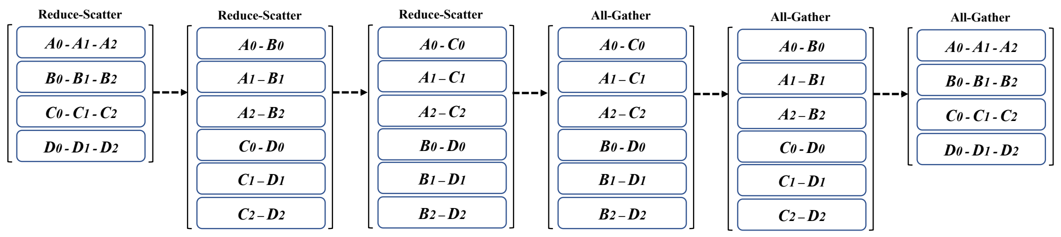
granularity nor flexibility sufficient enough to utilize the underlying hardware and the highly optimized implementations (i.e., ones offered by the hardware vendors) efficiently. We, however, found that the `reduce-scatter` and `all-gather` can be further decomposed into multiple stages of parallelizable `reduce-scatter` and `all-gather` operations in some symmetric cases. In detail, Fig. 3 shows that `all-reduce` can be first broken into one `reduce-scatter` followed by `all-gather` (the arrows indicate dependency). However, additional decomposition is possible if the following integer factorization exists:

$$|P| = p_0 p_1 p_2 ... p_k = \prod_{i<k} p_i \quad (p_i \in N, p_i > 1) \tag{8}$$

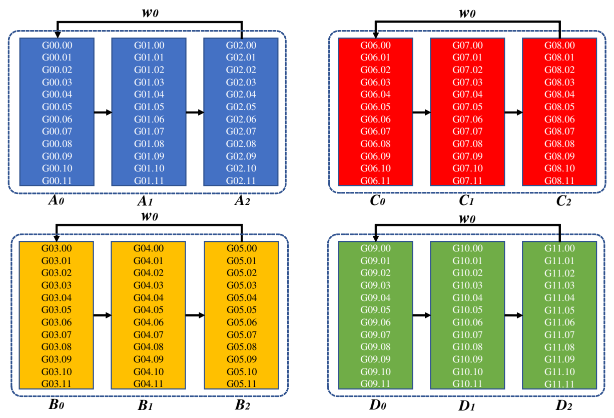
Then, the `reduce-scatter` can be further decomposed into $k-1$ stages of bundled `reduce-scatter` operations where the i-th stage has $\frac{P}{p_i}$ *concurrently* launchable `reduce-scatter` operations over different subsets of learners. The `all-gather` can also be further decomposed in the same 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:

- Decomposition can offer enough granularity and flexibility to map operations to underlying network elements and implementations.
- Higher parallelism at each stage can increase the bandwidth utilization (Sivakumar et al., 2000; Yildirim et al., 2016).

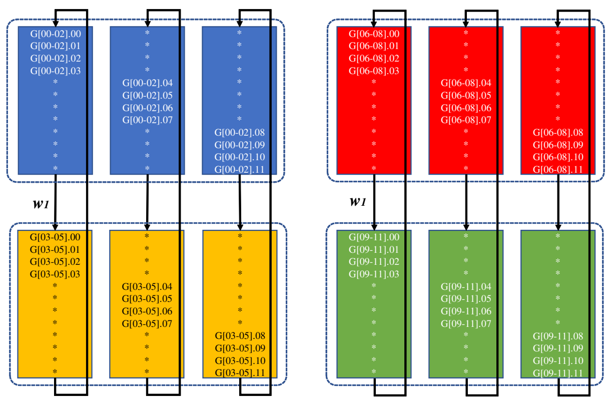
BlueConnect is essentially based on the proposed `all-reduce` decomposition in order to exploit network topology and corresponding software stacks better.



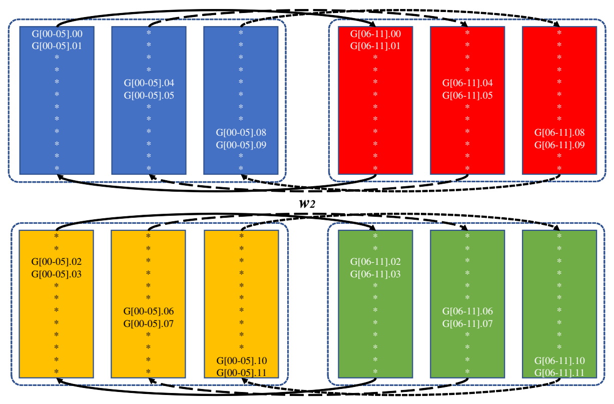
(a) `all-reduce` is decomposed into the multiple stages of `reduce-scatter` /`all-gather` operations.



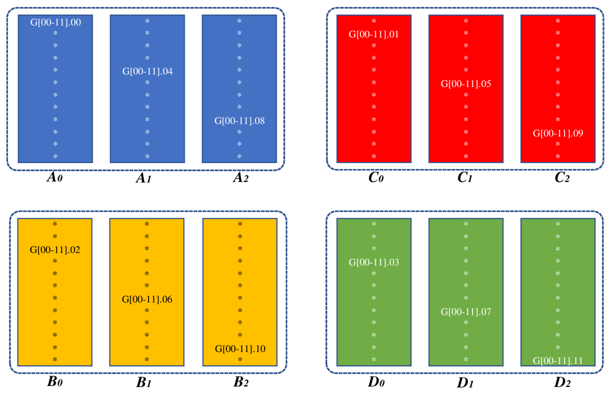
(b) step 1: 4 parallel `reduce-scatter` operations with w_0



(c) step 2: 6 parallel `reduce-scatter` operations with w_1



(d) step 3: 6 parallel `reduce-scatter` operations with w_2



(e) the final `reduce-scatter` result

Figure 4. BlueConnect `reduce-scatter` example for 12 GPUs with $|P| = p_0 p_1 p_2$ where $p_0 = 3, p_1 = 2$, and $p_2 = 2$. The reverse steps with `all-gather` shall be taken to complete `all-reduce`.

3.2 Algorithm

In this section, we describe BlueConnect algorithm. The key idea in BlueConnect is to decompose the synchronization or `all-reduce` of gradients across all learners into multiple/concurrent `reduce-scatter` and `all-gather` operations based on Section 3.1, then map them to the under-lying network fabrics. Therefore, BlueConnect has a decomposition step which can be done offline based on the network topology as in Section 3.2.1 and `all-reduce` step which shall be executed online following the decomposition as in Section 3.2.2. The pseudo code of BlueConnect on a learner is presented and explained in Section 3.2.3.

3.2.1 *Decomposition*

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 in Eq. (8)). Then, the amount of parallelism in each stage is determined by the number of elements in each network hierarchy level. Fig. 4 (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. (5)). Note that arrays to `all-reduce` are partitioned and notated as $G[a - b].c$ which represents a partially reduced result over the learners from a to b (inclusive) at the partition index c in Fig. 1. For example, $G[00 - 05].05$ represents the reduced results across the learners $\{0, 1, 2, 3, 4, 5\}$ with respect to the partition index 5.

3.2.2 *All-Reduce*

Once decomposition is completed, BlueConnect executes `reduce-scatter` and `all-gather` operations on various partitions of the input data, in a MPI-compliant manner, which will be explained in Section 3.2.3. Considering `all-reduce` for Fig. 1, BlueConnect performs the following steps:

Fig. 4 (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. 4 (c): Six short `reduce-scatter` operations are performed concurrently with w_1. $A_{\{0,1,2\}} \to B_{\{0,1,2\}}$ run over $s_{1.0}$, while $C_{\{0,1,2\}} \to D_{\{0,1,2\}}$ run over $s_{1.1}$, all concurrently. Note that data size for each instance is $\frac{N}{3}$.

Fig. 4 (d): Six independent `reduce-scatter` operations, $A_{\{0,1,2\}} \to C_{\{0,1,2\}}$ and $B_{\{0,1,2\}} \to D_{\{0,1,2\}}$ are performed concurrently over $s_{2.0}$ with w_2, yet the data size for each instance is only $\frac{N}{6}$.

Fig. 4 (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. 4, 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). $A_{\{0,1,2\}} \to B_{\{0,1,2\}}$ run over w_1 concurrently in Fig. 4 (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. 4 (a), six disjoint sets of ring communication patterns, $A_{\{0,1,2\}} \to C_{\{0,1,2\}}$ and $B_{\{0,1,2\}} \to D_{\{0,1,2\}}$ 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 $\frac{w_2}{6}$).

3.2.3 *Pseudo Code*

In this section, we describe BlueConnect in Algorithm 1 and its implementation details in the MPI context. Algorithm 1 assumes that topology-aware decomposition can be described by a utility like the rank file (OpenMPI) so that $|P|$ has been decomposed according to W. Then for a given gradient $G[N]$ and a global rank r (MPI-Tutorial), BlueConnect performs the one-time preparation step in lines 2-5. For a network switch type i, line 3 obtains a set of learners which will work with a local learner r over different bandwidths. For instance, $c(A_1, w_0) = \{A_0, A_1, A_2\}$ yet $c(A_1, w_1) = \{A_1, B_1\}$ in Fig. 4 (a). Then, line 4 computes the local rank of the current learner r among $c[i]$. As described in Fig. 4, BlueConnect performs a series of concurrent `reduce-scatter` followed by `all-gather` collectives which is in lines 7-16. Both `reduce-scatter` and `all-gather` operate on $G[g : g + \frac{N}{n})$ on a communicator $c[i]$. While moving up the network topology, the

$$\begin{aligned}
T_{blc} &= 2\mathcal{T}_{r/c}(p_0, N, \alpha, w_0) + 2\mathcal{T}_{r/c}(p_1, \frac{N}{p_0}, \alpha, \min\{w_0, \frac{w_1}{p_0}\}) + 2\mathcal{T}_{r/c}(p_2, \frac{N}{p_0 p_1}, \alpha, \min\{w_0, \frac{w_1}{p_0}, \frac{w_2}{p_0 p_1}\}) \\
&\quad + ... + 2\mathcal{T}_{r/c}(p_{|W|-1}, \frac{N}{\prod_{j=0}^{|W|-2} p_j}, \alpha, \min_{0 \le j < |W|}\{\frac{w_j}{\prod_{k=0}^{j-1} p_k}\}) \\
&= 2 \sum_{i=0}^{|W|-1} \mathcal{T}_{r/c}(p_i, \frac{N}{\prod_{j=0}^{i-1} p_j}, \alpha, \min_{0 \le j < i}\{\frac{w_j}{\prod_{k=0}^{j-1} p_k}\}) \qquad (9)
\end{aligned}$$

size of the `reduce-scatter` problem decreases with a growing n, and the gradient offset g is adjusted accordingly. Then, in the reverse order as in line 12, the size of the `all-gather` problem grows with a decreasing n, and the gradient offset g is adjusted accordingly as well. Since Algorithm 1 is for one learner and all other learners perform the same procedure with a different global rank r, BlueConnect keeps all learners busy and leaves no idle hosts unlike the two-level scheme.

3.3 Performance Model

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. (9). We can easily prove that BlueConnect offers smaller latency than the two-level scheme in Eq. (7).

We also show a BlueConnect performance model on torus topology which is another popular network topology and could be cheaper than fat-tree scheme (Solnushkin, 2013). The key advantage of using BlueConnect on torus is that it would reduce the bandwidth sharing on inter-node communication, as torus has dedicated connections between hosts. Assuming p_0 is the number of learners within a node, we can obtain the following BlueConnect performance model on torus.

$$T_{blc} = 2 \sum_{i=0}^{|W|-1} \mathcal{T}_{r/c}(p_i, \frac{N}{\prod_{j=0}^{i-1} p_j}, \alpha, \min_{1 \le j < i}\{w0, \frac{w_j}{p_0}\}) \qquad (10)$$

Note that Eq. (6, 7) are still valid on torus, as both run a single communication stream which will be bottlenecked by the most narrow bandwidth.

Algorithm 1 $BlueConnect(G[N], P = \prod_{j=0}^{|W|-1} p_j)$

```
1: $r$ = global-rank()
2: for $i \in 0 : |W| - 1$ do
3:     $c[i] = c(r, w_i)$
4:     $l[i]$ = local-rank($r, c[i]$)
5: end for
6: $n = 1, g = 0$
7: for $i \in 0 : |W| - 1$ do
8:     reduce-scatter($g, \frac{N}{n}, c[i]$)
9:     $n = n \times p_i$
10:    $g = g + \frac{N}{n} l[i]$
11: end for
12: for $i \in |W| - 1 : 0$ do
13:    $n = n \div p_i$
14:    $g = g - \frac{N}{n} l[i]$
15:    all-gather($g, \frac{N}{n}, c[i]$)
16: end for
```

3.4 Limitations

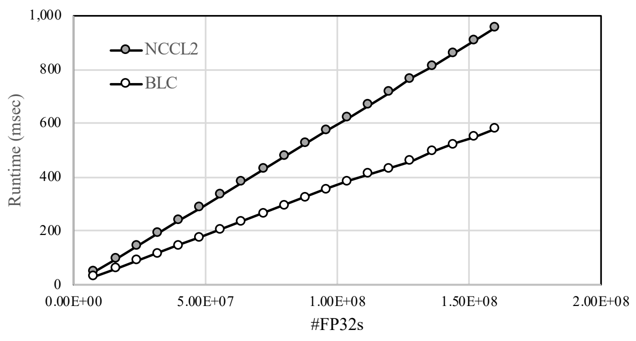
BlueConnect highly relies on `all-reduce` decomposition, thus if there is no feasible case for Eq (8), BlueConnect gets degenerated into a simple one-level ring scheme. However, considering the reality that all the hosts have the same number of GPUs over a symmetric network topology in most cases, BlueConnect can deliver high-performance `all-reduce` in practice.

4 Experimental Results

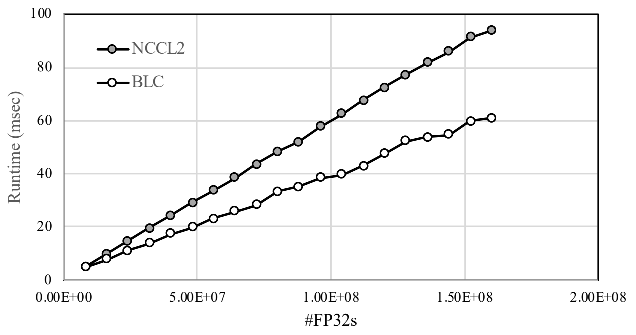
We implemented BlueConnect (**BLC**) for GPU in C++ based on CUDA-aware MPI (IBM, 2017b) and NCCL ver. 2 (NVidia, 2017a) (without using `all-reduce` APIs) to exchange gradients efficiently. BLC picks the best performing `reduce-scatter` and `all-gather` implementation directly from MPI and NCCL, or from custom implementations. We performed two sets of experiments to study the efficiency of **BLC**.

4.1 BLC compared with NCCL2

In this section, we report the pure `all-reduce` performance comparison between **BLC** and NCCL (i.e., ncclAllReduce) on two different setups. In one setup, we used two Intel Xeon(R) CPU E5-2680 systems with 4 Nvidia Telsa P100-PCIE-16GB GPUs each, connected through 10Gbps Ethernet. Within the Intel systems, the GPUs are connected through PCIe gen3. In the other setup, we used two IBM S822LC systems with 4 NVidia Tesla P100-SXM2 GPUs each, connected through 100Gbps InfiniBand. Within the IBM systems, the GPUs are connected through NVLink (IBM, 2017a; NVLink, 2017). Fig. 5 shows that



(a) Intel systems with PCIe gen3 and 10Gbps Ethernet



(b) IBM systems with NVLink and 100Gbps InfiniBand

Figure 5. `All-reduce` performance on two systems.

BLC outperforms NCCL by exploiting the network hierarchy within systems as well as between systems on both setups, over a wide range of FP32 floating-point number counts. Thanks to the faster network in the IBM platform, both **BLC** and NCCL perform about 10x faster than the Intel platform, but **BLC** is about 1.6x faster than NCCL on both cases.

4.2 BLC integrated in Caffe2

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. Each rack was equipped with a rack switch that was connected to a director switch. ImageNet-1K benchmark has been preloaded onto RAM Disk on each system to prevent performance degradation due to disk I/O. We compared **BLC** with the following deep learning communication techniques/libraries under the identical environment:

MPI_Allreduce `all-reduce` function in MPI (Thakur et al., 2005) which is optimized for generic communication of various sizes/topologies.

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

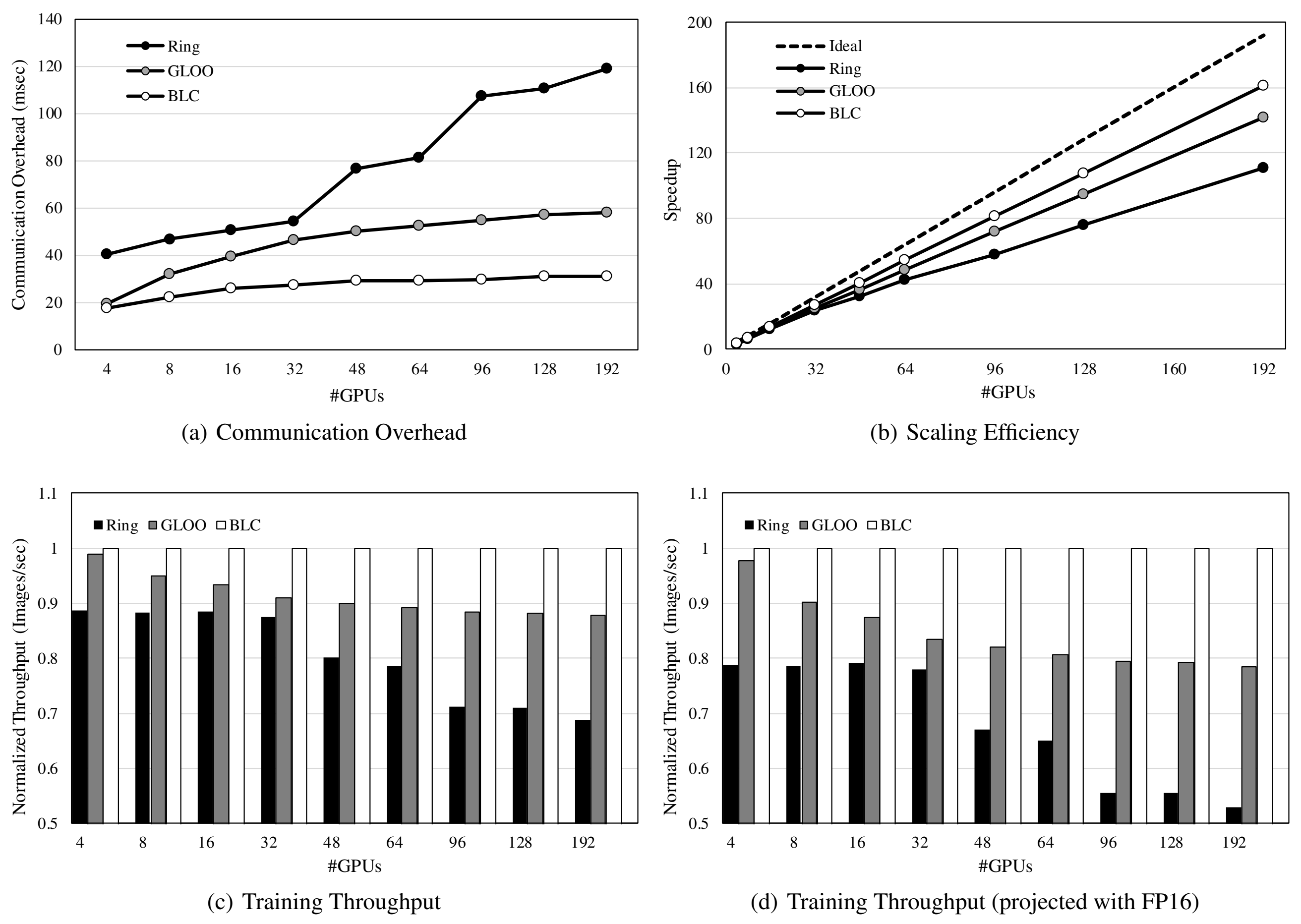
GLOO Two-level `all-reduce` algorithm in (Facebook, b; Jia et al., 2018), designed for deep learning based on NCCL (NVidia, 2017a) and ib_verb.

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 do not focus on the convergence/accuracy in this paper, as all three techniques compute `all_reduce` results synchronously and accurately. Nevertheless, we confirmed that our **BLC** integration into Caffe2 does not alter the convergence behavior through several tests.

We present our results in Fig. 6 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. 6 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.



(a) Communication Overhead

(b) Scaling Efficiency

(c) Training Throughput

(d) Training Throughput (projected with FP16)

Figure 6. Training performance comparison over 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**.

- (d) presents the projected throughput with FP16 on the future generation GPUs (i.e., Ampere GPUs) by scaling down the single-GPU performance by 4.8 (2x faster than Volta GPUs (NVidia, 2017b)) and cutting the communication overhead by half. It indicates that the throughput gap between **BLC** and others would get wider (e.g., from 11% to 22% on 192 GPUs compared with **GLOO**), supporting our claim that a faster communication algorithm is crucial for deep learning on more powerful computing resources.

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. To proliferate this technology to the masses, more research needs to be done, because massive GPU scaling relies on successful training to good accuracy for large batch size. Prior techniques such as (Goyal et al., 2017; You et al., 2017a) have been demonstrated on some neural network types, but we need to extend it to other popular neural network types, in particular, recurrent neural networks. The whole training has to be made resilient and elastic since it is very likely that some devices will malfunction when the number of devices increases. Automation and usability issues have to be addressed to enable more turnkey operation, especially in a cloud environment.

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.

Almási, George, Heidelberger, Philip, Archer, Charles J., Martorell, Xavier, Erway, C. Chris, Moreira, José E., Steinmacher-Burow, B., and Zheng, Yili. Optimization of mpi collective communication on bluegene/l systems. In *Proceedings of the 19th Annual International Conference on Supercomputing*, ICS '05, pp. 253–262, 2005. ISBN 1-59593-167-8.

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, Jianmin, Monga, Rajat, Bengio, Samy, and Józefowicz, Rafal. Revisiting distributed synchronous SGD. *CoRR*, abs/1604.00981, 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.

Iandola, Forrest N., Ashraf, Khalid, Moskewicz, Matthew W., and Keutzer, Kurt. Firecaffe: near-linear acceleration of deep neural network training on compute clusters. *CoRR*, abs/1511.00175, 2015.

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

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

Ioffe, Sergey and Szegedy, Christian. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pp. 448–456, 2015.

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.

Jouppi, Norman P., Young, Cliff, Patil, Nishant, Patterson, David, Agrawal, Gaurav, Bajwa, Raminder, Bates, Sarah, Bhatia, Suresh, Boden, Nan, Borchers, Al, Boyle, Rick, Cantin, Pierre-luc, Chao, Clifford, Clark, Chris, Coriell, Jeremy, Daley, Mike, Dau, Matt, Dean, Jeffrey, Gelb, Ben, Ghaemmaghami, Tara Vazir, Gottipati, Rajendra, Gulland, William, Hagmann, Robert, Ho, Richard C., Hogberg, Doug, Hu, John, Hundt, Robert, Hurt, Dan, Ibarz, Julian, Jaffey, Aaron, Jaworski, Alek, Kaplan, Alexander, Khaitan, Harshit, Koch, Andy, Kumar, Naveen, Lacy, Steve, Laudon, James, Law, James, Le, Diemthu, Leary, Chris, Liu, Zhuyuan, Lucke, Kyle, Lundin, Alan, MacKean, Gordon, Maggiore, Adriana, Mahony, Maire, Miller, Kieran, Nagarajan, Rahul, Narayanaswami, Ravi, Ni, Ray, Nix, Kathy, Norrie, Thomas, Omernick, Mark, Penukonda, Narayana, Phelps, Andy, Ross, Jonathan, Salek, Amir, Samadiani, Emad, Severn, Chris, Sizikov, Gregory, Snelham, Matthew, Souter, Jed, Steinberg, Dan, Swing, Andy, Tan, Mercedes, Thorson, Gregory, Tian, Bo, Toma, Horia, Tuttle, Erick, Vasudevan, Vijay, Walter, Richard, Wang, Walter, Wilcox, Eric, and Yoon, Doe Hyun. In-datacenter performance analysis of a tensor processing unit. *CoRR*, abs/1704.04760, 2017.

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.

Keuper, Janis. Distributed training of deep neuronal networks: Theoretical and practical limits of parallel scalability. *CoRR*, abs/1609.06870, 2016.

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

Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, pp. 1097–1105. 2012.

Lian, Xiangru, Zhang, Ce, Zhang, Huan, Hsieh, Cho-Jui, Zhang, Wei, and Liu, Ji. Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent. 2017.

MPI-Tutorial. http://mpitutorial.com/tutorials/introduction-to-groups-and-communicators.

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

Niu, Feng, Recht, Benjamin, Re, Christopher, and Wright, Stephen J. Hogwild!: A lock-free approach to parallelizing stochastic gradient descent. In *Proceedings of the 24th International Conference on Neural Information Processing Systems*, NIPS'11, pp. 693–701, 2011.

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. 2017a.

NVidia. https://devblogs.nvidia.com/parallelforall/inside-volta. 2017b.

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

OpenMPI. https://www.open-mpi.org/projects/hwloc.

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.

Solnushkin, Konstantin S. Automated design of torus networks. *CoRR*, abs/1301.6180, 2013.

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.

Wang, Jianyu and Joshi, Gauri. Cooperative sgd: A unified framework for the design and analysis of communication-efficient sgd algorithms. 2018.

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.

Zhang, Sixin, Choromanska, Anna, and LeCun, Yann. Deep learning with elastic averaging sgd. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'15, pp. 685–693, 2015a.

Zhang, Wei, Gupta, Suyog, Lian, Xiangru, and Liu, Ji. Staleness-aware async-sgd for distributed deep learning. *CoRR*, abs/1511.05950, 2015b.