Highly Scalable Deep Learning Training System with Mixed-Precision: Training ImageNet in Four Minutes

Xianyan Jia[*1], **Shutao Song**[*1], **Wei He**[1], **Yangzihao Wang**[1], **Haidong Rong**[1], **Feihu Zhou**[1], **Liqiang Xie**[1], **Zhenyu Guo**[1], **Yuanzhou Yang**[1], **Liwei Yu**[1], **Tiegang Chen**[1], **Guangxiao Hu**[1], **Shaohuai Shi**[*2], **Xiaowen Chu**[2]

Tencent Inc.[1], Hong Kong Baptist University[2]

{xianyanjia, sampsonsong, winsonwhe, slashwang, hudsonrong, hopezhou, felixxie, alexguo, jokeyang, leolwyu, steelchen, gethinhu}@tencent.com;

{csshshi, chxw}@comp.hkbu.edu.hk

[*]Authors contributed equally

ABSTRACT

Synchronized stochastic gradient descent (SGD) optimizers with data parallelism are widely used in training large-scale deep neural networks. Although using larger mini-batch sizes can improve the system scalability by reducing the communication-to-computation ratio, it may hurt the generalization ability of the models. To this end, we build a highly scalable deep learning training system for dense GPU clusters with three main contributions: (1) We propose a mixed-precision training method that significantly improves the training throughput of a single GPU without losing accuracy. (2) We propose an optimization approach for extremely large mini-batch size (up to 64k) that can train CNN models on the ImageNet dataset without losing accuracy. (3) We propose highly optimized all-reduce algorithms that achieve up to 3x and 11x speedup on AlexNet and ResNet-50 respectively than NCCL-based training on a cluster with 1024 Tesla P40 GPUs. On training ResNet-50 with 90 epochs, the state-of-the-art GPU-based system with 1024 Tesla P100 GPUs spent 15 minutes and achieved 74.9% top-1 test accuracy, and another KNL-based system with 2048 Intel KNLs spent 20 minutes and achieved 75.4% accuracy. Our training system can achieve 75.8% top-1 test accuracy in only **6.6 minutes** using 2048 Tesla P40 GPUs. When training AlexNet with 95 epochs, our system can achieve 58.7% top-1 test accuracy within **4 minutes**, which also outperforms all other existing systems.

KEYWORDS

Deep Learning, Synchronized SGD, Mixed-Precision, Large Mini-batch Size, All-Reduce

1 INTRODUCTION

With the ever-increasing sizes of datasets and larger deep neural networks, training often takes several days if not weeks (For example, training ResNet-50 [13] takes 29 hours using 8 Tesla P100 GPUs). Extremely long training time impedes the research and development progress. Due to the single machine's limited computing resources, it is natural to distribute the workload to clusters and use supercomputing power to increase the throughput of data flow. A commonly adopted solution is distributed synchronous Stochastic Gradient Descent (SGD) which parallelizes the tasks across machines. To make full use of the hardware, mini-batch size per machine should be properly set and cannot be too small. In addition, it is common to use large batch to achieve weak scaling [2, 4, 5, 12, 23, 27]. In this way, the speedup is obtained by utilizing overall throughput of the system and fewer updates of the model.

However, there are two challenges when using large batch across large clusters:

- **Challenge 1**: Larger mini-batch size often leads to lower test accuracy, as there exists a generalization gap [15].
- **Challenge 2**: When using large clusters, it is harder to achieve near-linear scalability as the number of machines increases, especially for models with the high communication-to-computation ratio.

Challenge 1. Larger mini-batch size reduces the variance of gradients by taking the average of the gradients in mini-batch, and it provides a more accurate estimate of the gradients [11]. Thus it allows the model to take bigger step size and in turn makes the optimization algorithm progress faster. However, as reported in [27], when increasing the mini-batch size to 64K, the test accuracy of ResNet-50 drops from 75.4% to 73.2%. Codreanu et al. [5] also train ResNet-50 with batch 64K and achieves the accuracy of 73.9%, which does not meet the baseline accuracy.

Challenge 2. A distributed training system with data parallelism strategy typically divides batches across each GPU, and requires a gradient aggregation step between each training step. This communication step usually becomes the bottleneck of the system when the number of GPUs becomes large. To achieve high performance for such distributed training system, we need to improve both the single GPU performance and the overall system performance. Given that the training throughput with one GPU is S, if we use N GPUs with the scaling efficiency e, then the system throughout should be $T = S \cdot N \cdot e$. When the number of GPUs N is fixed, we need to increase both S and e to improve the overall throughput of the training system T. To improve the throughput, we need faster computation and more efficient bandwidth utilization, to improve the scaling efficiency, we need more efficient collective communication primitives that can handle a system with thousands of GPUs.

In this paper, we have addressed the above two challenges. Our contributions are as follows:

- We successfully scale the mini-batch size to 64K for AlexNet and ResNet-50 training without loss of accuracy. To achieve this, we have adopted and proposed several strategies (i.e.,

mixed-precision training with LARS, eliminated weight decay on bias and parameters for batch normalization, and adding proper batch normalization layers).
- We build a high-throughput distributed deep learning training system which contains two main features to improve the single GPU performance S and the system scaling efficiency e. 1) To improve S, our system supports half-precision training, which theoretically, could achieve two times throughput improvement compared to its single-precision counterpart. 2) To improve e, our system uses a hybrid strategy which combines our optimized adaptive all-reduce collective with the ring-based all-reduce in NCCL.

The rest of this paper is structured as follows. We show the related work in Section 2, then describe the design and implementation of our system and optimizations in Section 3 and Section 4. Finally, we discuss our experimental results in Section 5 and conclude with experiences we have learned through building such a system in Section 6.

2 RELATED WORK

This section describes the research landscape of distributed deep learning training system in three fields: 1) large-batch training; 2) low-precision training; 3) distributed training on heterogeneous clusters.

2.1 Large-batch Training

Goyal et al. [12] first trains the ResNet-50 ImageNet model with a large mini-batch size of 8K over 256 Tesla GPUs and finishes the training process within one hour. They adopt the linear scaling rule to adjust the learning rate as a function of mini-batch size. They also develop a warmup scheme, i.e., starting from small learning rate and slowly increasing the learning rate through a few epochs, in order to overcome the optimization challenges in the first few epochs. Cho et al. [4] use the same training configuration and finish ResNet-50 training in 50 minutes with 256 GPUs. You et al. [27] further increase the mini-batch size of ResNet-50 from 8K to 32K. They use LARS to enable large mini-batch size and can finish the ResNet-50 training in 20 minutes with 2048 KNL chips. Besides ResNet-50, they also experiment on AlexNet and finish the training of mini-batch size 32K on ImageNet in 11 minutes with 1024 Skylake CPUs. Akiba et al. [2] demonstrate the training of ResNet-50 in 15 minutes with a mini-batch size of 32K over 1024 Tesla P100 GPUs. They adopt techniques such as RMSprop warm-up, batch normalization without moving average and a slow-start learning rate schedule. However, their reported test accuracy is 74.9%, which is lower than the baseline ~75.3%. Codreanu et al. [5] use a combination of techniques such as aggressive learning rate scheduling and improved weight decay. They reach 75.3% test accuracy using 32K mini-batch size in 42 minutes and 74.6% test accuracy using 49K mini-batch size in 28 minutes. In addition to adjusting the learning rate, Smith et al. [23] propose to increase the mini-batch size instead of decaying the learning rate. Their work is the first to train ImageNet with less time (30 minutes) without losing accuracy after Goyal et al. [12]. All the above research towards large-batch training either fails to scale to more nodes and more GPUs with larger mini-batch size, or trade accuracy loss for better performance.

Devarakonda et al. [9] use dynamic mini-batch size and decay the learning rate at the same time. However, adapting the mini-batch size is only tested in piece-wise constant learning rate schedule, but cannot be easily applied in polynomial decay, whose curve is more smooth.

2.2 Low-precision Training

Low-precision computation is often used to lower the time and energy cost of machine learning. Unfortunately, the benefits of low-precision (LP) arithmetic come with a cost. The round-off or quantization error that results from converting numbers into a low-precision representation introduces noise that can affect the convergence rate and accuracy of SGD. Conventional wisdom says that, for training, low-precision introduces a tradeoff of the number-of-bits used versus the statistical accuracy: the fewer bits used, the worse the solution will become. Theoretical upper bounds on the performance of low-precision SGD [9] and empirical observations of implemented low-precision algorithms [6] further confirm that current algorithms are limited by this precision-accuracy tradeoff. De Sa et al. [7] describe a simple low-precision stochastic gradient descent variant called HALP, which converges at the same theoretical rate as full-precision algorithms despite the noise introduced by using low precision throughout execution. The key idea is to use Stochastic Variance Reduced Gradient (SVRG) to reduce gradient variance, and to combine this with a novel technique called bit centering to reduce quantization error. Micikevicius et al. [19] propose three techniques for preventing the loss of critical information. Firstly, they recommend maintaining a single-precision copy of weights that accumulates the gradients after each optimizer step (this copy is rounded to half-precision for the forward- and back-propagation). Secondly, they propose loss-scaling to preserve gradient values with small magnitudes. Although the loss-scaling technique was not a requirement for successful mixed-precision training when mini-batch size is not large enough. Thirdly, they use half-precision arithmetic that accumulates into single-precision outputs, which are converted to half-precision before storing to memory. While all tensors in the forward and backward passes were in FP16 format, a master copy of weights was updated in FP32 format. However, they have not applied this with large-batch training strategy such as LARS to achieve better performance.

2.3 Distributed Training on Heterogeneous Clusters

Most distributed machine learning frameworks such as TensorFlow [18] adopt centralized deployment mode. One bottleneck of the centralized algorithm is the high communication cost on the central nodes. Baidu [10] first introduced the ring-based all-reduce algorithm [3] to deep learning. This is a very important contribution to the field of distributed training. The ring all-reduce algorithm greatly reduces the communication load when the number of nodes increases. However, the original version is low in bandwidth utilization because of splitting up the tensors data into too small slices when tensor sizes are small compared to the number of nodes in the cluster. The IBM's PowerAI Distributed Deep Learning (DDL) system [4] has mentioned a new all-reduce algorithm. However,

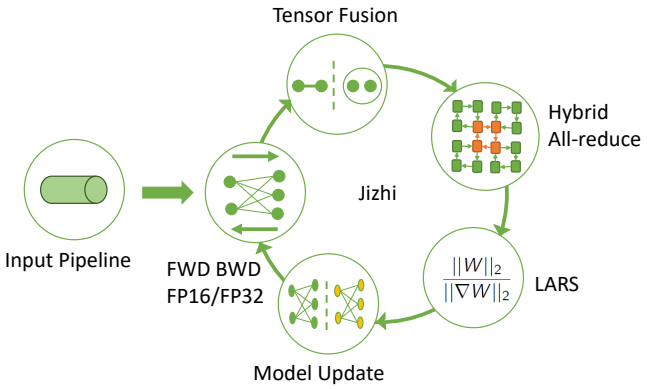

Figure 1: Jizhi Training System Overview

since the implementation is closed source, it is difficult to be applied to other works. Goyal et al. [12] use an implementation of all-reduce consists of three phases for intra-node and inter-node communication: 1) intra-node reduce, 2) inter-node all-reduce, and 3) intra-node broadcast, which reduces the communication load across nodes and improves scalability. Horovod [20] introduced gradient fusion strategy to the all-reduce algorithm, which reduces tensor fragmentation and improves bandwidth utilization. However, this indiscriminate fusion results in unnecessary memory copy and no significant gains have been made in our test scenario. The DAG model proposed by Shi et al. [22] for scheduling the computation and communication tasks in synchronized SGD guides us to design our optimized all-reduce algorithm.

Our system adopts several useful parts from the above work. Together with other optimizations, they help us yield high scalability on ImageNet training with both AlexNet and ResNet-50.

3 SYSTEM OVERVIEW

Figure 1 is an overview of our distributed deep learning training system. At a high level, our system contains the following three modules: 1) input pipeline module; 2) training module; and 3) communication module.

- The input pipeline module delivers data for the next step before the current step has finished. It uses pipelining in order to minimize both CPU and GPU idle time.
- The training module includes model construction and variable management. In this module, we have incorporated optimizations such as forward/backward computation with mixed-precision and model update with LARS.
- The communication module uses tensor fusion and hybrid all-reduce to optimize the scaling efficiency according to tensor size and cluster size.

4 SYSTEM IMPLEMENTATION AND OPTIMIZATIONS

4.1 Mixed-Precision Training with LARS

As Micikevicius et al. [19] have mentioned, the motivation of using half-precision (FP16) in the training phase is to lower memory bandwidth pressure as well as increase arithmetic throughput. The former can be achieved by using fewer bits to store the same number of values, the latter is achieved on processors that offer higher throughput for reduced precision math. Orthogonal to half-precision training, You et al. [26] first proposed LARS to enable larger mini-batch size for distributed training. The algorithm introduces a local learning rate for each layer (as shown in Equation 1), which is the ratio of the L2-norm of weights and gradients weighted by a LARS coefficient η. Gradients are multiplied with its adaptive local learning rate. A natural choice is to combine half-precision training with LARS to achieve larger mini-batch size with scalability. However, a naïve implementation would introduce several problems because using LARS directly on half-precision training will cause the computed learning rate to be out of the dynamic range of IEEE half-precision format (FP16), and thus cause the gradients to vanish and stall the training process.

$$\Delta w_t^l = \gamma \cdot \eta \cdot \frac{\|w^l\|}{\|\nabla L(w^l))\|} \cdot \nabla L(w_t^l) \tag{1}$$

To cope with this situation, we have proposed a training strategy which uses mixed-precision training with LARS as shown in Figure 2. In our strategy, the operations in forward and backward propagation are performed in FP16, while the weights and gradients are cast to single-precision (FP32) format before applying LARS and cast back to FP16 afterward.

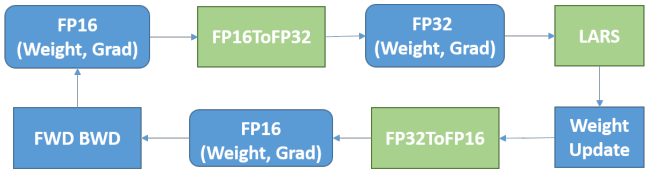

Figure 2: Mix-precision training with LARS

Mixed-precision training with LARS is one of the critical reasons that our system could keep good scalability while increasing the mini-batch size to 64K. Table 1 shows that on ResNet-50 with the mini-batch size of 64K, using LARS with mixed-precision training could maintain the top-1 accuracy as 76.2%.

Table 1: Effectiveness of using LARS on ResNet-50

Mini-Batch Size	Number of Epochs	LARS	Top-1 Accuracy
64K	90	NO	73.2%
64K	90	YES	76.2%

4.2 Improvements on Model Architecture

Improvements in model architecture could often lead to better performance. In our system, we have improved the model architecture from the following two aspects: 1) eliminated weight decay on the bias and batch normalization; and 2) adding proper batch normalization layers for AlexNet.

Weight decay is a commonly-used strategy to get better generalization for the model by adding a regularization term to the loss

function E [17].

$$E(w) = E_0(w) + \frac{1}{2}\lambda \sum_i w_i^2 \tag{2}$$

If gradient descent is used for learning, the last term of the loss function leads to a new term $-\lambda w_i$ in the gradients update:

$$w_i^{t+1} = w_i^t - \eta \frac{\partial E}{\partial w_i^t} - \lambda w_i^t \tag{3}$$

In neural network training, it is a typical practice to penalize only the weights of the affine transformation at each layer and leaves the biases unregularized [11]. What we have observed in our training for AlexNet is that if we also leave two parameter sets β and γ in batch normalization unregularized, our model could achieve better convergence and usually with less time to train for the same number of epochs. β and γ are two trainable parameters in batch normalization as shown in below formulas, where $\mu_{\mathcal{B}}$ is the mean of mini-batch and σ^2 is the variance of mini-batch. β and γ contrals the scale and shift of the normalized result.

$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2 + \epsilon}}$$

$$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv BN_{\gamma,\beta}(x_i)$$

As Goodfellow et al. [11] has noted, the reason that our model achieves better convergence could be that b, β and γ usually have much less parameters compared to weights W (for AlexNet model, b, β and γ parameters only amount to 0.02% of all parameters), which means that leaving them unregularized would not give too much variance and regularizing could instead, introduce a significant amount of underfitting. As shown in Table 2, for AlexNet, we get around 1.3% improvement in accuracy with the same number of epochs. The slight improvements of training run time come from the reduced computations in L2 regularization.

Table 2: Effect of Regularization with b, β and γ for AlexNet

Batch	Epochs	Regularize b, β and γ	Top1
64K	95	Yes	55.8%
64K	95	No	57.1%

As mentioned in LARS [26], replacing Local Response Normalization layers with Batch Normalization(BN) could improve the accuracy of AlexNet [14]. However, as shown in Table 2, such AlexNet-BN model cannot reach baseline top-1 accuracy when the mini-batch size increases to 64K. By analyzing the parameters and feature map distributions, we find that the feature map distribution after Pool5 has a larger variance and maximum values as training go on (as shown in Figure 4(a)). The significant change of feature scaling makes the training difficult. This motivates us to insert another BN layer after Pool5 to rescale the feature map as shown in Figure 3. The refined-AlexNet-BN model could reach 58.8% top-1 accuracy with 64K mini-batch size for 95 Epoch training.

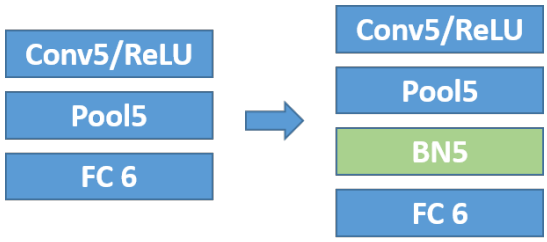

Figure 3: Insert Batch Norm after Pool5 for AlexNet

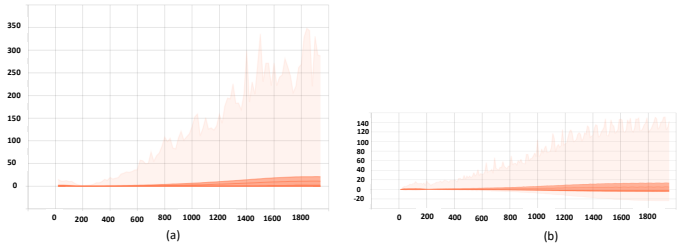

Figure 4: Feature Map Distribution of Pool5(a) and Pool5-BN5(b) of AlexNet as shown in Figure 3. (the horizontal axis is the training steps, the vertical axis is the feature map distributions.)

4.3 Improvements on Communication Strategies

For large batch training with distributed synchronized SGD, efficient gradients aggregation across all GPUs after each iteration is crucial to the training performance [25][21]. Goyal et al. [12] have pointed out that for models with a larger set of parameters and GPUs with more computing power, it becomes harder to hide the cost of aggregation in the backprop phase. In our case, for the mini-batch size of 64K and 1024 GPUs, gradients aggregation using collective communication primitives such as all-reduce has become the bottleneck of the system. NCCL 2.0 is optimized for dense multi-GPU systems such as NVIDIA's DGX-1. In our case, communication happens over a hundred of nodes on a cluster, the traditional ring-based all-reduce implementation does not scale due to the following reason: In a cluster with k GPUs, Ring all-reduce will split the data on each GPU into k chunks and do the reduce in $k-1$ iterations [24]. When k gets larger, the messages passing between nodes will become smaller and fail to utilize the full bandwidth of the network. To cope with this problem, we have developed two strategies:

- *Tensor Fusion.* An efficient communication strategy in a distributed training system should maximize the throughput as well as reduce the latency. The main challenge of training deep neural networks with multiple layers is that the sizes of gradient tensors to aggregate vary a lot for different types of layers. Usually, gradient tensor sizes for convolution layers are much smaller than fully-connected layers. Sending too many small tensors in the network will not only cause the bandwidth to be under-utilized but also increase the latency. To cope with this problem, we adopt the technique of tensor

fusion. The core idea of tensor fusion is to pack multiple small size tensors together before all-reduce to better utilize the bandwidth of the network. We set a parameter θ. In the backward phase, as tensors from each layer come in, we fuse them into a buffer pool if the total size is less than θ, and only send the fused tensor out for all-reduce when the total size is larger than θ. This strategy could be easily generalized to distributed training for other neural networks. Figure 6 and Figure 7 shows the fusion strategy for AlexNet and ResNet-50 respectively.

- *Hierarchical All-reduce.* In our experiments for ResNet-50, when using tensor fusion to combine all the tensors into a single tensor, the end-to-end performance will increase by 8x. However, the high throughput also increases the latency, since fusing into a single tensor will prevent the parallelization of gradient aggregation of last few layers and backward propagation of earlier layers. To reduce latency, we need to restrict the condition for tensor fusion, i.e. allow smaller, and multiple tensors in tensor fusion phase. However, ring all-reduce perform worse on small tensors. Hierarchical all-reduce could solve this problem for small tensor communication. Instead of using ring all-reduce where each GPU sends and receives $\frac{m}{p}$ bytes of data in $2(p-1)$ steps. We can group k GPUs together, then use a three-phase algorithm to do the all-reduce across all GPUs (Figure 5: first we do a reduce within GPUs of the same group, store the partial results to a master GPU in each group, then we launch Ring all-reduce across p/k groups, after each master GPU gets the final result, we do a broadcast within each group to propagate the final result to every GPU. The three-phase algorithm reduces the running steps from $2(p-1)$ to $4(k-1)+2(p/k-1)$ since the intra-group reduce and broadcast each costs $2(k-1)$ steps. The decrease of computation steps makes the three-phase algorithm perform better in latency-sensitive case (i.e. for small tensor size and the large number of GPUs). We set k as a tunable parameter and observe the highest performance is achieved when k is set to 16 in our 1024 GPU cluster.

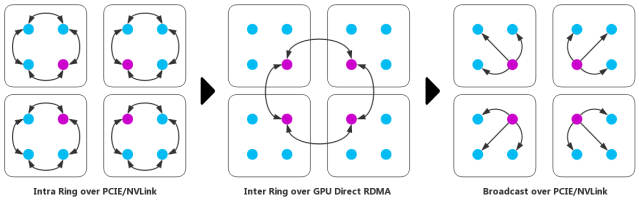

Figure 5: Our three-phase all-reduce algorithm for multi-node multi-GPU gradient aggregation.

- *Hybrid All-reduce.* Hierarchical all-reduce can bring performance gain for convolution layers which usually have a smaller number of weights. However, for fully-connected layers which usually have a much larger number of weights, ring-based all-reduce still outperforms our hierarchical all-reduce. To enjoy the best of both worlds, we use a hybrid strategy in our system. We set a parameter η to represent the size of the tensor to aggregate in bytes. By tuning this parameter, we can switch between the traditional ring-based all-reduce and our customized all-reduce. Combined with tensor fusion, hybrid all-reduce could help us achieve better performance.

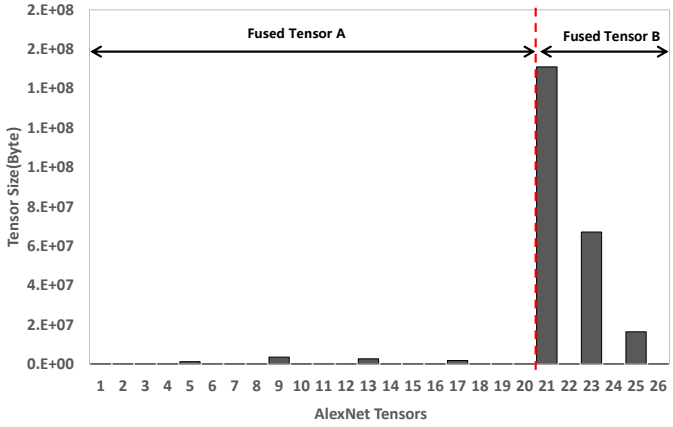

Figure 6: Tensor Fusion of AlexNet

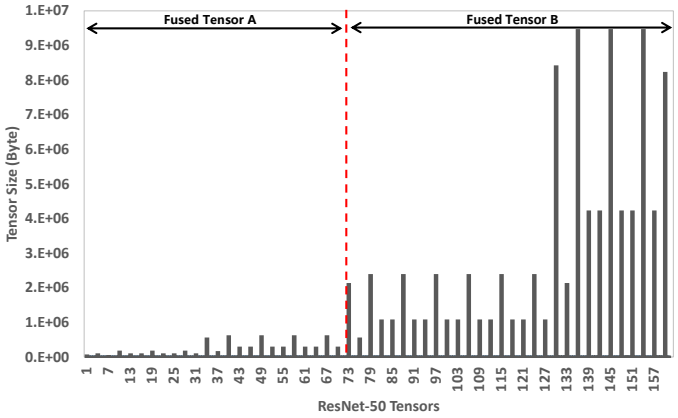

Figure 7: Tensor Fusion of ResNet-50

5 EXPERIMENTAL RESULTS

5.1 Experimental Settings

Model. We choose AlexNet [16] and ResNet-50 [13] for our experiments because they represent two typical types of CNN: As shown in Table 3, the parameter size of AlexNet is around 2.5 times as ResNet-50, while the computation of ResNet-50 is around 5.6 times as AlexNet. Thus, the bottleneck of AlexNet lies in communication, while the bottleneck of ResNet-50 lies in computation. The baseline top-1 accuracy of AlexNet is 58.8% [27] and the baseline top-1 accuracy of ResNet-50 is 75.3% [13].

Dataset. We use ImageNet [8] dataset in the following experiments. Both models are trained on 1.28 million training images and

Table 3: Model Information

Model	Input Size	Parameter Size	FLOPs	Baseline Top1
AlexNet	227x227	62M	727 M	58.8%
ResNet-50	224x224	25M	4 G	75.3%

evaluated on 50,000 validation images by top-1 test accuracy for its 1000 classes task. The training images are partitioned into 1024 chunks and the validation images are partitioned into 128 chunks. Images are stored in the format of TFRecord. In all our experiments, we use data augmentation offered in TensorFlow.

Software. We use TensorFlow [1] as a training framework for its flexible design, various use cases, and a large user/developer community. We build our distributed gradient aggregation algorithm with NVIDIA Collective Communication Library (NCCL), and OpenMPI.

Hardware. Our GPU cluster includes 256 nodes, and each node contains 8 NVIDIA Tesla P40 GPUs that are interconnected with PCIe. For local storage, each server has two 2T NVMe SSDs. For network connectivity, each server has a Mellanox ConnectX-4 100Gbit Ethernet network card. We use RoCEv2 (RDMA over Converged Ethernet) for communications among nodes in cluster, which is a common Remote Direct Memory Access (RDMA) implementations [1]. We also use GPUDirect RDMA (GDR) to enable direct data exchange between GPUs on different nodes. All of these technologies can reduce the latency and increase the scaling efficiency in our cluster.

5.2 Overall experimental results

For ResNet-50, as shown in Table 5, our system finishes the training in only 8.7 minutes with 76.2% top-1 accuracy over 1024 Tesla P40 GPUs and 6.6 minutes with 75.8% top-1 accuracy over 2048 Tesla P40 GPUs, which to our best knowledge is the state-of-the-art for ImageNet training. Compared to Akiba et al. [2], our work saves around 40% cost with similar hardware but much shorter training time. Compared to He et al. [13]'s work which uses 8 GPUs, we achieve more than 248x speedup. Based on the same 1024 GPUs, our work is 1.61 times faster than Akiba et al. [2]. Note that for ResNet-50 training, we adopt half-precision communication during the all-reduce gradients aggregation phase due to its reduced memory usage.

For AlexNet, previously, You et al. [27] could finish the ImageNet training with 32K mini-batch size in 11 minutes. As shown in Table 4, we break this record and finish the AlexNet training in 4 minutes with 1024 Tesla P40 GPUs.

5.3 Convergence Analysis

In this subsection, we show that with our optimizations, we can maintain the same convergence as previous works on ImageNet training with a larger mini-batch size. The overall training curve of top-1 accuracy for ResNet-50 and AlexNet are shown in Figure 8 and Figure 9 separately.

[1]RDMA is a technology which supports zero-copy networking by enabling the network adapter to transfer data directly to or from application memory, eliminating the need to copy data between application memory and the data buffers in the operating system.

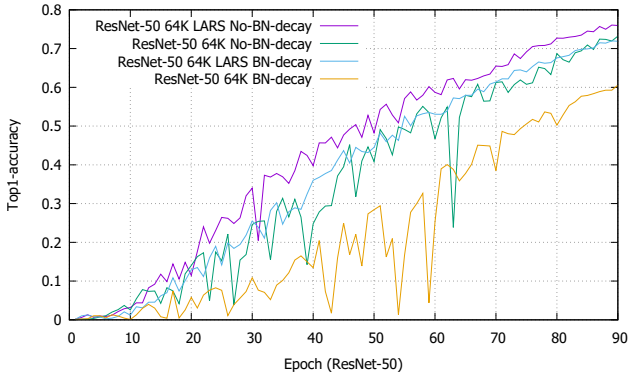

Figure 8: ImageNet Training with ResNet-50 Using 64K Mini-Batch Size

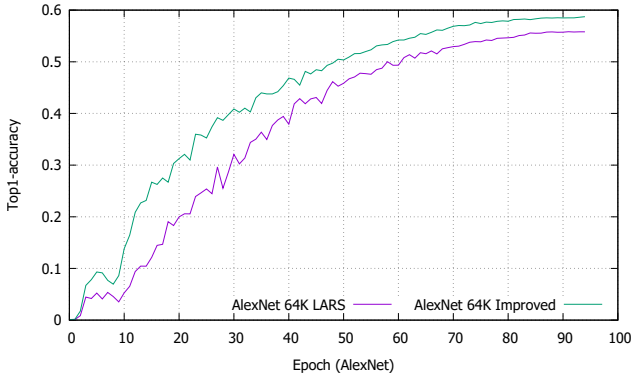

Figure 9: ImageNet Training with AlexNet Using 64K Mini-Batch Size

Compare the convergence of mixed-precision training and single-precision training. As explained in Section 4.1, we adopt a mixed-precision training strategy to avoid the precision loss in half-precision computation. A master copy of weights was updated in FP32 to avoid loss of accuracy, and all tensors in the forward and backward passes were in FP16. The updated gradients (weight gradients multiplied by the learning rate) become too small to be represented in FP16 as any value whose magnitude is smaller than 2^{-24} becomes zero in FP16. In fact, when the mini-batch size is less than 16K, a master copy of FP32 is enough to get the same top-1 test accuracy of baseline. With the mini-batch size of 16K, loss-scaling is required to maintain the same accuracy as the baseline, or else gradients vanishing will start to appear. When the mini-batch size increases to 32K, LARS technique was required for successful mixed precision training. To make LARS perform properly, we have to set its coefficient to a small number: 0.001. This will cause the local learning rate to become zeroes in FP16. Because of this, we need to assign an FP32 copy to LARS. Also, in order to avoid overfitting, the weight decay should be increased from 0.0001 to 0.0005 when the mini-batch size grows to 64K. To validate the effectiveness of our mixed-precision training strategy, we compare it with plain single-precision training. The experiment result in Figure 10 shows that

Table 4: Compare AlexNet training with different teams

Team	Batch	Hardware	Software	Top-1 Accuracy	Time
You et al. [27]	512	DGX-1 station	NVCaffe	58.8%	6h 10m
You et al. [27]	32K	CPU × 1024	Intel Caffe	58.6%	11min
This work	**64K**	Tesla P40 × 512	TensorFlow	**58.8%**	**5m**
This work	**64K**	Tesla P40 × 1024	TensorFlow	**58.7%**	**4m**

Table 5: Compare ResNet-50 training with different teams

Team	Batch	Hardware	Software	Top-1 Accuracy	Time
He et al. [13]	256	Tesla P100 × 8	Caffe	75.3%	29h
Goyal et al. [12]	8K	Tesla P100 × 256	Caffe2	76.3%	1h
Cho et al. [4]	8K	Tesla P100 × 256	Torch	75.0%	50min
Codreanu et al. [5]	32K	KNL × 1024	Intel Caffe	75.3%	42min
You et al. [27]	32K	KNL × 2048	Intel Caffe	75.4%	20min
Akiba et al. [2]	32K	Tesla P100 × 1024	Chainer	74.9%	15min
This work	**64K**	Tesla P40 × 1024	TensorFlow	**76.2%**	**8.7m**
This work	**64K**	Tesla P40 × 2048	TensorFlow	**75.8%**	**6.6m**

our mixed-precision training strategy has similar top-1 accuracy for ResNet-50 at 90 epoch as single-precision training (76.3% for single-precision training and 76.2% for mixed-precision training).

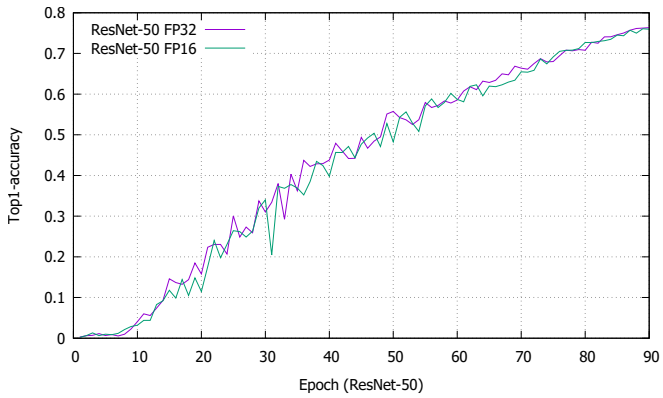

Figure 10: Compare the convergence of mixed-precision and single-precision training

Effect of LARS. We compare the test accuracy of ResNet-50 with and without applying LARS [26]. As shown in Table 6, LARS could improve the top-1 accuracy from 60.6% to 71.9%. Also, Figure 8 shows that with LARS, the training curve is more smooth than the curve without LARS. However, even with both mixed-precision and LARS, we still cannot reach the baseline accuracy yet.

Effect of model improvements. Eliminating weight decay on bias and batch normalization generates positive effects on convergence. Table 7 shows that eliminated weight decay on batch normalization(BN) for ResNet-50, combined with mixed-precision and LARS, could improve top-1 accuracy from 71.9% to 76.2%, which meets the baseline test accuracy. Note that for ResNet-50 training,

Table 6: Effect of LARS to ResNet-50 Training

Batch	LARS	Top-1 Accuracy
64K	×	60.6%
64K	✓	71.9%

we ignore the bias tensor for weight decay as its influence is negligible.

Table 7: Effect of improvements to ResNet-50 Training

Batch	No Decay BN	Top1
64K	×	71.9%
64K	✓	76.2%

Table 8: Effect of improvements to AlexNet Training

Batch	No Decay Bias	No Decay BN	pool5 BN	Top-1 Accuracy
64K	×	×	×	55.8%
64K	×	✓	×	56.3%
64K	✓	×	×	56.4%
64K	✓	✓	×	57.1%
64K	✓	✓	✓	58.8%

For AlexNet, we test the effect of optimization strategies including not regularizing bias, not regularizing batch norm parameters and inserting batch normalization after Pool5 layer. As shown in Table 8, when applying all strategies, the top-1 accuracy of mini-batch size 64K reaches its peak value of 58.8%, which meets the baseline accuracy. Figure 9 also shows that after applying a series

of optimization strategies, the convergence speed gets improved and the final test accuracy is higher than using the LARS algorithm only.

5.4 Training Speed and Scalability

In this subsection, we show the training speed and scalability of our distributed training system.

Compare the speed of mixed-precision training and single-precision training. As shown in Table 9, using mixed-precision training can speedup single-node performance of ResNet-50 from 172 images/second to 218 images/second. This improvement comes from the FP16 computation speedup and the reduced communication parameter size.

Table 9: ResNet-50: Compare the speed of mixed-precision training and single-precision training

Batch/GPU	Data Type	Images/Sec
64	FP32	172
64	mixed	218

Scalability. Figure 11 shows that our customized all-reduce has high scaling efficiency. When per GPU mini-batch size is fixed to 64, the scaling efficiency of 1024 GPUs (8 GPUs /*product* 128 nodes) compared to single-node (8 GPUs) could reach 99.2%, which is close to the optimal scalability. When comparing the scaling efficiency before and after optimization, we can see the improvements is significant. For 1024 GPUs, we improved the scaling efficiency from 9.0% to 99.2%.

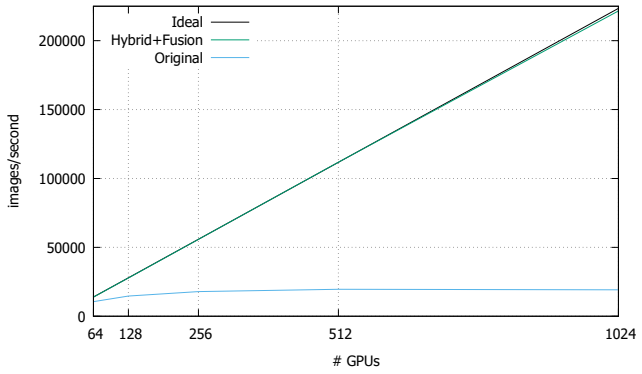

Figure 11: ResNet-50 training throughput with batch 64/GPU

When per GPU mini-batch size is fixed to 32, it is harder to scale out. Because the smaller mini-batch size often leads to faster computation, which causes the communication to become the bottleneck. As reported in [2], the scaling efficiency of 1024 GPUs with 32 batch/GPU is 80.0%. As shown in Figure 12, our system can reach 87.9% for the same batch settings as [2]. Due to our efficient communication strategies, we have achieved higher scaling efficiency than the state-of-the-art with the same mini-batch size.

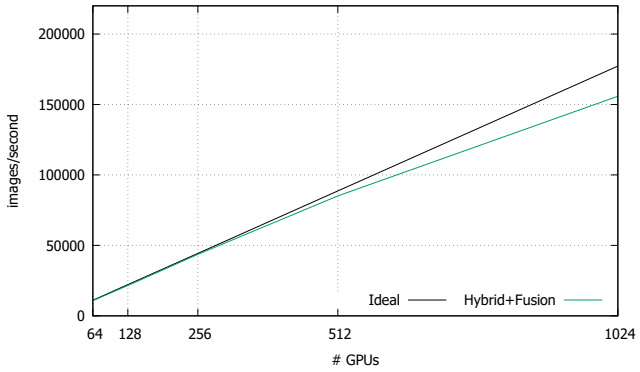

Figure 12: ResNet-50 training throughput with batch 32/GPU

Figure 13 shows the scalability of AlexNet training with a mini-batch size of 128 per GPU. The baseline is FP32 all-reduce (with RDMA). When comparing the scaling efficiency between using 8 GPUs and 512 GPUs, introducing tensor fusion could achieve an improvement from 70% to 81%, and using FP16 all-reduce gives 82.7% scalability. When combining FP16 and tensor fusion strategies with hybrid all-reduce, we get **91.4%** scaling efficiency.

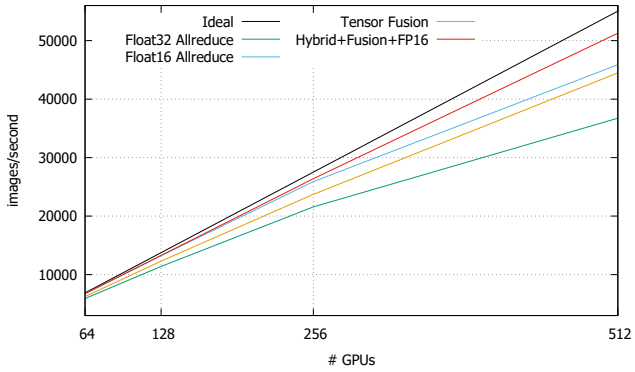

Figure 13: AlexNet training throughput with batch 128/GPU

6 CONCLUSION

Large-scale deep neural networks require a large amount of computation to converge to a good testing accuracy. Synchronized gradient descent methods with data parallelism are widely used to train the models in distributed environments. However, data communication between machines in the cluster easily becomes the bottleneck of the system throughput. Though using a large mini-batch size can improve the scalability of the system, it becomes more difficult to keep the good generalization ability of the models. In this study, we build a highly scalable deep learning training system to address the problem. We first use the mixed-precision techniques to improve the throughput of a single GPU without losing accuracy. Then we propose optimization approaches (e.g., eliminated weigh decay in batch normalization layers) to successfully train AlexNet and

ResNet-50 using a mini-batch size of 64K without losing accuracy. To further increase the scalability of the system, we propose highly optimized all-reduce algorithms which achieve much better performance than the NCCL-based counterpart. As a result, on the training of the ImageNet dataset, we achieve 58.7% top-1 test accuracy with AlexNet (95 epochs) in only 4 minutes using 1024 Tesla P40 GPUs, and achieve 75.8% top-1 test accuracy with ResNet-50 (90 epochs) in only 6.6 minutes using 2048 Tesla P40 GPUs, which outperforms the existing systems.

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, and others. 2016. Tensorflow: a system for large-scale machine learning.. In *OSDI*, Vol. 16. 265–283.

[2] Takuya Akiba, Shuji Suzuki, and Keisuke Fukuda. 2017. Extremely large minibatch sgd: Training resnet-50 on imagenet in 15 minutes. *arXiv preprint arXiv:1711.04325* (2017).

[3] M. Barnett, L. Shuler, R. van de Geijn, S. Gupta, D. G. Payne, and J. Watts. 1994. Interprocessor collective communication library (InterCom). In *Proceedings of IEEE Scalable High Performance Computing Conference*. 357–364. https://doi.org/10.1109/SHPCC.1994.296665

[4] Minsik Cho, Ulrich Finkler, Sameer Kumar, David Kung, Vaibhav Saxena, and Dheeraj Sreedhar. 2017. PowerAI DDL. *arXiv preprint arXiv:1708.02188* (2017).

[5] Valeriu Codreanu, Damian Podareanu, and Vikram Saletore. 2017. Scale out for large minibatch SGD: Residual network training on ImageNet-1K with improved accuracy and reduced time to train. *arXiv preprint arXiv:1711.04291* (2017).

[6] Matthieu Courbariaux, Y Bengio, and Jean-Pierre David. 2015. Training deep neural networks with low precision multiplications. (12 2015).

[7] Christopher De Sa, Megan Leszczynski, Jian Zhang, Alana Marzoev, Christopher R Aberger, Kunle Olukotun, and Christopher Ré. 2018. High-Accuracy Low-Precision Training. *arXiv preprint arXiv:1803.03383* (2018).

[8] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. Ieee, 248–255.

[9] Aditya Devarakonda, Maxim Naumov, and Michael Garland. 2017. AdaBatch: Adaptive Batch Sizes for Training Deep Neural Networks. *arXiv preprint arXiv:1712.02029* (2017).

[10] Andrew Gibiansky. 2017. Bringing HPC techniques to deep learning. http://research.baidu.com/bringing-hpc-techniques-deep-learning

[11] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. 2016. *Deep learning*. Vol. 1. MIT press Cambridge.

[12] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. 2017. Accurate, large minibatch SGD: training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677* (2017).

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[14] Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167* (2015).

[15] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. 2016. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836* (2016).

[16] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. 1097–1105.

[17] Anders Krogh and John A. Hertz. 1992. A Simple Weight Decay Can Improve Generalization. In *Advances in Neural Information Processing Systems 4*, J. E. Moody, S. J. Hanson, and R. P. Lippmann (Eds.). Morgan-Kaufmann, 950–957. http://papers.nips.cc/paper/563-a-simple-weight-decay-can-improve-generalization.pdf

[18] Jianmin Chen Zhifeng Chen Andy Davis Jeffrey Dean Matthieu Devin Sanjay Ghemawat Geoffrey Irving Michael Isard Manjunath Kudlur Josh Levenberg Rajat Monga Sherry Moore Derek G. Murray Benoit Steiner Paul Tucker Vijay Vasudevan Pete Warden Martin Wicke Yuan Yu Xiaoqiang Zheng Martin Abadi, Paul Barham. 2015. TensorFlow: A system for large-scale machine learning. *12th USENIX Symposium on Operating Systems Design and Implementation* (2015).

[19] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaev, Ganesh Venkatesh, and others. 2017. Mixed precision training. *arXiv preprint arXiv:1710.03740* (2017).

[20] Alexander Sergeev and Mike Del Balso. 2018. Horovod: fast and easy distributed deep learning in TensorFlow. *arXiv preprint arXiv:1802.05799* (2018).

[21] Shaohuai Shi and Xiaowen Chu. 2017. Performance Modeling and Evaluation of Distributed Deep Learning Frameworks on GPUs. *arXiv preprint arXiv:1711.05979* (2017).

[22] Shaohuai Shi, Qiang Wang, Xiaowen Chu, and Bo Li. 2018. Modeling and Evaluation of Synchronous Stochastic Gradient Descent in Distributed Deep Learning on Multiple GPUs. *arXiv preprint arXiv:1805.03812* (2018).

[23] Samuel L Smith, Pieter-Jan Kindermans, and Quoc V Le. 2017. Don't Decay the Learning Rate, Increase the Batch Size. *arXiv preprint arXiv:1711.00489* (2017).

[24] Rajeev Thakur, Rolf Rabenseifner, and William Gropp. 2005. Optimization of collective communication operations in MPICH. *The International Journal of High Performance Computing Applications* 19, 1 (2005), 49–66.

[25] Pijika Watcharapichat, Victoria Lopez Morales, Raul Castro Fernandez, and Peter Pietzuch. 2016. Ako: Decentralised deep learning with partial gradient exchange. In *Proceedings of the Seventh ACM Symposium on Cloud Computing*. ACM, 84–97.

[26] Yang You, Igor Gitman, and Boris Ginsburg. 2017. Scaling SGD Batch Size to 32K for ImageNet Training. *CoRR* abs/1708.03888 (2017). arXiv:1708.03888 http://arxiv.org/abs/1708.03888

[27] Yang You, Zhao Zhang, C Hsieh, James Demmel, and Kurt Keutzer. 2017. ImageNet training in minutes. *CoRR, abs/1709.05011* (2017).