Communication-Efficient Local Stochastic Gradient Descent for Scalable Deep Learning

Sunwoo Lee, Qiao Kang, Ankit Agrawal, Alok Choudhary, and Wei-keng Liao
Department of Electrical and Computer Engineering
Northwestern University
Evanston, USA
{slz839,qkt561,ankitag,choudhar,wkliao}@eecs.northwestern.edu

***Abstract*—Synchronous Stochastic Gradient Descent (SGD) with data parallelism, the most popular parallel training strategy for deep learning, suffers from expensive gradient communications. Local SGD with periodic model averaging is a promising alternative to synchronous SGD. The algorithm allows each worker to locally update its own model, and periodically averages the model parameters across all the workers. While this algorithm enjoys less frequent communications, the convergence rate is strongly affected by the number of workers. In order to scale up the local SGD training without losing accuracy, the number of workers should be sufficiently small so that the model converges reasonably fast. In this paper, we discuss how to exploit the degree of parallelism in local SGD while maintaining model accuracy. Our training strategy employs multiple groups of processes and each group trains a local model based on data parallelism. The local models are periodically averaged across all the groups. Based on this hierarchical parallelism, we design a model averaging algorithm that has a cheaper communication cost than *allreduce*-based approach. We also propose a practical metric for finding the maximum number of workers that does not cause a significant accuracy loss. Our experimental results demonstrate that our proposed training strategy provides a significantly improved scalability while achieving a comparable model accuracy to synchronous SGD.**

***Index Terms*—Deep Learning, Local SGD, Parallel Training**

I. Introduction

Training a deep and large neural network is an extremely time-consuming task which can take hours or even days. To reduce the training time, researchers have put much effort into developing parallel training algorithms. The most popular parallel training algorithm is synchronous Stochastic Gradient Descent (SGD) with data parallelism. This parallel training strategy evenly distributes each mini-batch to all workers, and each worker independently processes the given training samples. Then, the locally computed gradients are averaged across all the workers using inter-process communications. Although this synchronous parallel approach guarantees a fast convergence of training loss, the scalability is limited due to the expensive gradient communications.

Recently, local SGD has been highlighted due to its less frequent communications than synchronous SGD. We refer the local SGD with periodic model averaging to 'local SGD' for short. The algorithm allows workers to locally train their own models and periodically averages the model parameters across all the workers. Thus, the communications are performed only when averaging the model parameters. Compared to the synchronous SGD that averages the gradients every iteration, local SGD enjoys a significantly reduced communication frequency, and thus a better scalability.

Despite the low communication frequency, local SGD has a limitation that hinders effective scaling such that the convergence rate of training loss is adversely affected as the number of workers increases. It also has been empirically shown that, given the same number of training epochs and a fixed local batch size, a larger number of workers yields a lower validation accuracy [1], [2]. For instance, Lin et. al observed a considerable accuracy drop when increased the number of workers to 16 [1]. To address this issue, the authors proposed *post-local SGD* that uses a single worker in the early epochs and then increases the number of workers once the learning rate is decayed. This approach significantly sacrifices the degree of parallelism. The experimental results in [3] also show that the training loss converges to a higher value as the number of workers increases. In practice, researchers use $8 \sim 16$ workers for local SGD training.

In this paper, we discuss how to exploit the degree of parallelism of local SGD without a significant accuracy loss. We first describe a hierarchical parallelism that applies data parallelism to each local model training. Our training strategy employs multiple groups of processes and each group independently trains a local model based on data parallelism. The local models are periodically averaged across all the groups. Thus, this hierarchical parallelism has two communications, one for averaging gradients within each group and the other for averaging model parameters across the groups. Usually, local SGD is implemented with *allreduce* operations such that each local model is updated using the gradients and then the model parameters are aggregated and summed up across all the processes using *allreduce* operations. In this approach, all the workers periodically average the entire model parameters, and thus the communications cost is a constant regardless of the number of processes. We propose a communication-efficient model averaging algorithm that allows each process to average only a distinct subset of model parameters across the groups. The communication cost of the proposed algorithm is reduced as more processes work in each group.

We also discuss how to find the proper number of workers for local SGD, that makes a good balance between the

978-1-7281-6251-5/20/$31.00 ©2020 IEEE

scalability and the convergence rate. We argue that the number of workers for local SGD should be considered as a tunable hyper-parameter since it strongly affects the training convergence rate as well as the generalization performance. We propose a metric for estimating how effectively the workers contribute to minimizing the shared cost function. Based on the metric, we explain how to find the largest number of workers that achieves a good convergence rate. The proposed distance metric enables to find the proper number of workers within a few early iterations rather than running the entire training.

We validate our proposed training strategy using popular benchmark classification and regression applications. First, we compare the learning curves between the synchronous SGD and local SGD with different numbers of workers. We empirically verify the effectiveness of the proposed distance metric by showing that the accuracy does not drop when the number of workers is chosen based on the proposed distance metric. Then, we present the strong scaling performance and analyze the impact of the proposed hierarchical parallelism with the communication-efficient model averaging algorithm on it. Our experimental results demonstrate that our local SGD-based parallel training strategy dramatically improves the scalability of neural network training without a significant accuracy loss with a minimal extra tuning effort.

II. Background and Related Works

We first define a few notations for describing the algorithms.

- B: the mini-batch size
- P: the number of workers
- I: the model averaging interval
- μ: the learning rate

A. Synchronous SGD with Data Parallelism

Synchronous SGD represents a synchronous-parallel version of mini-batch SGD. Mini-Batch SGD iteratively adjusts the model parameters using the gradients computed from a random subset of training samples that is called mini-batch. The algorithm updates the model parameters using Equation 1.

$$w_{t+1} = w_t - \mu \frac{1}{B} \sum_{i=1}^{B} \nabla f(w_t, x_i), \tag{1}$$

where w_t is the model parameters at iteration t, and $\nabla f(w_t, x_i)$ is the stochastic gradient of a cost function f with respect to the model parameters w_t, computed from a training sample x_i.

In synchronous SGD with data parallelism, mini-batch SGD is parallelized such that each mini-batch of size B is evenly distributed to all P workers and processed independently. Then, the workers synchronize their gradients by a global averaging operation. Finally, the shared model parameters are updated using the averaged gradients so that all P workers always view the same model parameters. Typically, the model parameters are averaged using *allreduce* operations. The communication cost of an *allreduce* operation is proportional to the data size. Thus, when the neural network consists of a large number of parameters, the parallel training suffers from the expensive communications and it results in achieving a poor scaling performance.

B. Local SGD with Periodic Model Averaging

Local SGD has been highlighted due to the lower communication frequency than the classical synchronous SGD. Given P workers, each worker trains its own model using Equation 1. After every I iteration, the model parameters are averaged among all the P workers using Equation 2.

$$\overline{w}_t = \frac{1}{P} \sum_{i=1}^{P} w_{t,i}, \tag{2}$$

where $w_{t,i}$ is the local model parameters of worker i at iteration t.

Recently, a few variants of local SGD have been proposed. Cong et. al [4] proposed Sparse Aggregation SGD (SASGD) which locally accumulates the gradients and periodically sums up the gradients among all the workers to adjust the shared model parameters. Lin et. al [1] proposed post-local SGD which employs the classical mini-batch SGD in the early training epochs. Yu et. al [3] named the same algorithm as restart SGD and studied the theoretical convergence analysis. The authors showed that the convergence rate is $O(\frac{1}{PT})$ assuming $T > P^3$, where T is the number of iterations. Haddadpour et. al [2] also analyzed the convergence rate of local SGD. The authors showed that the expected difference of the training loss after T iterations and the optimal training loss is bounded by $O(\frac{1}{PBT})$ when I is small enough. Although these works present slightly different conclusions, we can observe two common facts from their convergence rate analysis. First, to achieve a fast convergence rate, the model averaging interval should be sufficiently small. Some researchers proposed adaptive model averaging interval methods to improve the scaling efficiency [2], [5]. Second, the larger the number of workers, the slower the model converges. This is a critical limitation that hinders the scalable parallel training of neural network.

III. Communication-Efficient Local SGD for Parallel Deep Learning

In this section, we discuss our parallelization strategy for local SGD-based neural network training. We first describe a hierarchical parallelism that improves the degree of parallelism in local SGD training. Based on the hierarchical parallelism, we design a model averaging algorithm that has a cheaper communication cost than *allreduce*-based approach. Then, we discuss how to find the proper number of workers for local SGD training. Our tuning method is based on a distance metric that estimates how effectively the workers contribute to minimize the shared cost function.

A. Hierarchical Parallelism

Local SGD iteratively assigns disjoint B samples on each worker, and P workers independently process the given samples and update their local models every iteration. Thus, in data parallel training, the degree of parallelism of local SGD

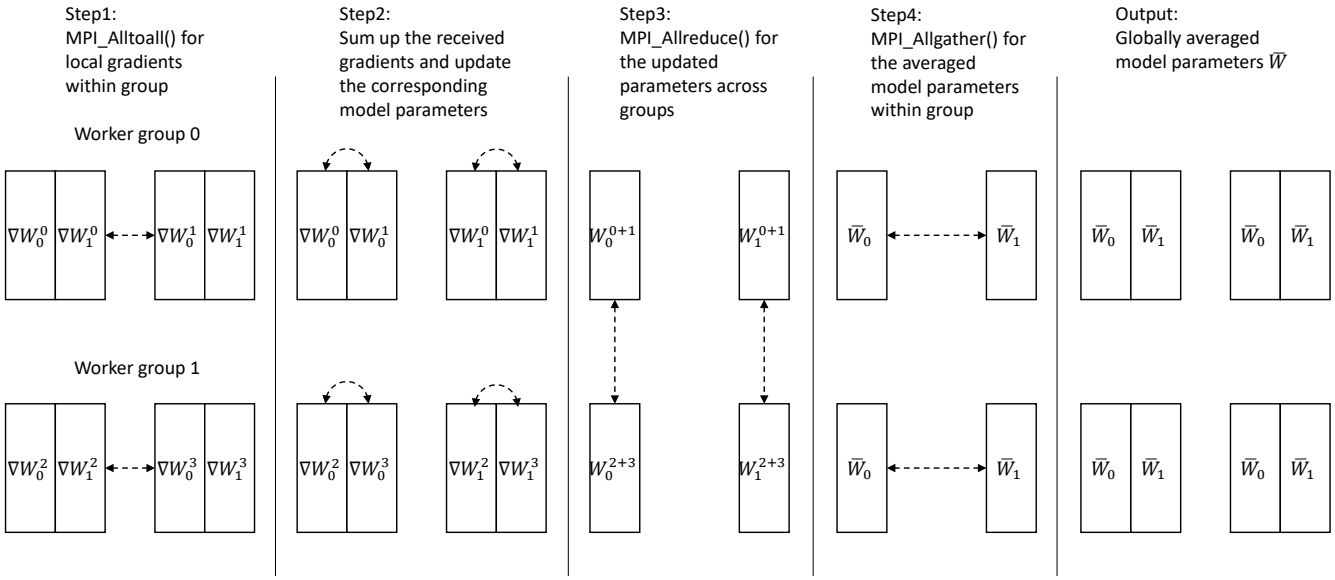

Fig. 1. An example of model averaging in local SGD training. Two worker groups run local SGD and each group employs data parallelism with 2 processes.

is BP. However, the typical setting in local SGD is that one process trains one model. For example, Wang et. al [2] run 4 processes each of which trains its own VGG/ResNet model on a separate GPU. Lin et. al [1] locally train 16 ResNet models using 16 processes. Yu et. al [3] run 8 pyTorch processes on 8 GPUs for VGG training.

Motivated by these observations, we design a hierarchical parallel training strategy that fully utilizes the degree of parallelism. Our training strategy consists of two levels of parallelism. First, K processes contribute to a single shared model based on data parallelism. Each mini-batch of size B is evenly distributed to the K processes and the gradients are independently calculated. Then, the locally computed gradients are averaged across the K processes and the shared model is updated by the averaged gradients. Second, P groups of such processes train their models independently and periodically average the model parameters across all the groups after every I iterations. The number of processes that participate in the training is PK in total. Note that, in our discussion, a 'worker' indicates each K processes that train one shared model.

The described hierarchical parallel training strategy has two inter-process communications. First, once a mini-batch is processed by each group of processes, the locally computed gradients are averaged across all the K processes within each group. Given N training samples, this communication is performed $\frac{N}{BP}$ times per epoch. Second, the model parameters are averaged across P groups after every I iterations. The model parameters are averaged $\frac{N}{BPI}$ times per epoch.

If the training is performed on accelerators such as GPUs or Intel Xeon Phi, the ratio of computation to communication is significantly reduced, and thus the communication time becomes the performance bottleneck. We overlap the gradient communications with the computations in order to improve the scaling efficiency of each local model training. Since the gradients do not have data dependency across layers, the gradient communication at one layer and the gradient computations at other layers can be performed simultaneously. Recently, many researchers have studied how to maximize the overlap of computation and communication [6]–[10]. We adopt the gradient averaging algorithm proposed in [10]. This communication algorithm enables to overlap the gradient communications with not only the backward computations but also the forward computations at the next iteration.

B. Communication-Efficient Model Averaging

In local SGD training, the communication cost for averaging the model parameters scales up with the number of workers. We propose a communication-efficient model averaging algorithm based on our proposed hierarchical parallelism. First, once every process computes the gradients, K processes within each group performs an *all-to-all* personalized communication for the local gradients. After the communication, each process becomes to have K sets of local gradients of a distinct subset of parameters. Second, each process element-wisely averages the received gradients and update the corresponding parameters. Third, each process averages the updated parameters by calling an *allreduce* communication across P groups and element-wisely multiplying $\frac{1}{P}$. Finally, all the processes perform an *allgather* communication within each group. After these 4 steps, every process ends up having a full set of the state-of-the-art parameters.

Figure 1 presents an example of the proposed communication algorithm. The example shows the case of 2 groups ($P = 2$) each of which has 2 processes ($K = 2$). In Figure 1, ∇W_j^i indicates the gradients computed by process i for the j^{th} subset of the parameters. After 3 communication and 1 computation steps, all the individual process has a full set of parameters $\bar{W}^{0\sim3}$.

Communication Cost Analysis – We analyze and compare the communication cost between the classical *allreduce*-based approach and our proposed algorithm. In this work, we follow the cost model used in many previous works [11]–[13],

$$T = s\alpha + w\beta, \tag{3}$$

where s is the number of messages, w is the message size, α is the communication latency cost, and β is the reciprocal bandwidth.

In all the previous local SGD works, *Allreduce* operations were used for averaging the model parameters across workers [1]–[3], [5]. MPICH implements the *Allreduce* operation using two different algorithms depending on the message size [12], [14]. When the message is smaller than or equal to 2KB, a binomial tree algorithm is used, which is known to be optimal. For the messages larger than 2KB, Rabenseifner's algorithm is used. In practice, the size of gradients at one layer in modern neural networks is most likely larger than 2KB. So, we consider the communication cost of Rabenseifner's algorithm only in this work. The communication cost of the algorithm is as follows [12].

$$2log(p)\alpha + 2\frac{p-1}{p}w\beta, \tag{4}$$

where p is the number of processes that participate in the communication.

There are two communications in our hierarchical parallelism; one for averaging the gradients among K processes within each group and the other for averaging the model parameters across P worker groups. So, when *MPI_Allreduce* operation is used for these two communication, the overall communication cost within each epoch is calculated as follows.

$$\begin{aligned} T = &\frac{N}{BP}(2log(K)\alpha + 2\frac{K-1}{K}M\beta) + \\ &\frac{N}{BPI}(2log(K)\alpha + 2\frac{P-1}{P}M\beta), \end{aligned} \tag{5}$$

where N is the number of training samples, M is the number of parameters, B is the mini-batch size per worker group, P is the number of worker groups, K is the number of processes per worker group, and I is the model averaging interval. Note that, in this cost analysis, we consider that B, P, K, and I are all pre-defined constants.

The proposed model averaging algorithm consists of three separate communications. The first communication step is *MPI_Alltoall* and its communication cost T_0 is as follows.

$$T_0 = \frac{N}{BP}((K-1)\alpha + M\frac{K-1}{K}\beta) \tag{6}$$

The second communication step is *MPI_Allreduce* across the groups and its cost T_1 is as follows.

$$T_1 = \frac{N}{BPI}(2log(P)\alpha + \frac{2M}{K}\frac{P-1}{P}\beta) \tag{7}$$

Finally, the third communication step is *MPI_Allgather* and its cost T_2 is as follows.

$$T_2 = \frac{N}{BP}((K-1)\alpha + M\frac{K-1}{K}\beta) \tag{8}$$

In this work, we assume the startup time term of the cost model shown in Equation 3 is much smaller than the transfer time term. So, when comparing the communication cost, we focus on the second term only. Under this assumption, $T_0 + T_2$ is the same as the communication cost of a single *MPI_Allreduce* call with a message size of M, that is the first term of the right-hand side of Equation 5. So, the difference between T and $T_0 + T_1 + T_2$ comes from the message size of the *MPI_Allreduce* that is performed across the groups. In *allreduce*-based approach, the message size is M, the size of the entire model parameters, while that is $\frac{M}{K}$ in our approach. This analysis proves that our approach has a cheaper communication cost than *allreduce*-based approach when $K > 1$.

Applying to variants of local SGD – There have been several variants of local SGD such as *Post-local* SGD [1], Sparse Aggregation SGD (SASGD) [4], and a few adaptive averaging interval methods [2], [5]. Our proposed hierarchical parallelism and the model averaging method can be applied to all these algorithms because the proposed methods are independent of any parameter update rules or hyper-parameter settings. For instance, the *post-local* SGD begins the training with a single synchronous model and then increases the number of workers allowing local updates after the learning rate is decayed for the first time. Once the number of workers increases, it becomes exactly the same as the local SGD training, and thus our hierarchical parallelism and the model averaging algorithm can be directly applied to achieve a better scaling efficiency. SASGD averages the accumulated gradients across all the workers using *allreduce* communications. Likewise, if the hierarchical parallelism is applied to the training, the proposed communication pattern can be used to average the accumulated gradients instead, and thus a better scaling efficiency can be expected.

C. Hyper-Parameter Tuning for Scalable Local SGD

We discuss three hyper-parameters of local SGD, that affect the model accuracy: the local batch size B, the number of workers P, and the model averaging interval I. Especially, to the best of our knowledge, the number of workers P has not been considered as a hyper-parameters in the previous works, and set to a certain small number between $4 \sim 16$. In practice, P is usually equal to the number of available GPUs.

Finding the proper number of workers – We argue that the number of P is a hyper-parameter and should be carefully tuned to achieve a good convergence accuracy. In local SGD, the model moves on the parameter space when either the model parameters are updated using the gradients or the models are averaged across all the workers. The number of parameter updates is a constant which is the same as the number of mini-batches. However, local SGD averages the local models $\frac{N}{BPI}$ times per epoch. Thus, the number of workers P strongly affects how much the model can move on the parameter space in each epoch. In ideal, if the distance between the local models and the averaged model is proportionally increased as P increases, we can expect the

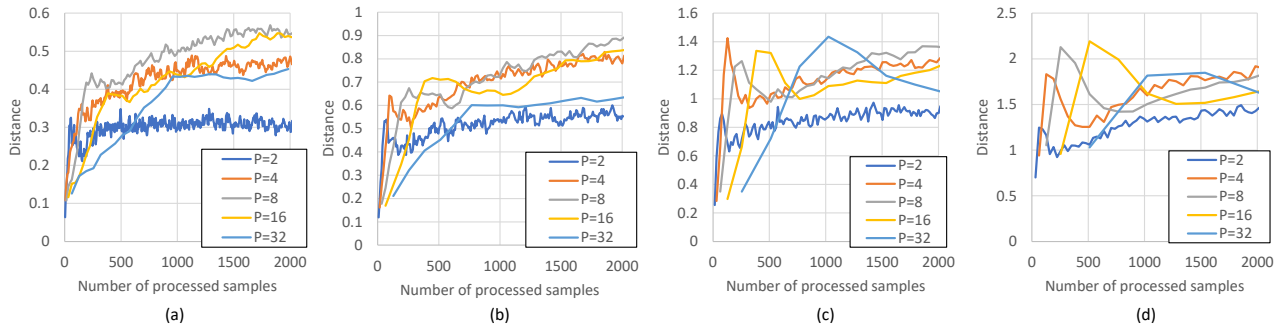

Fig. 2. The distances between local models and the averaged model with different numbers of workers and averaging intervals for CIFAR10 classification. The local batch size (B) is fixed to 128 and the number of workers (P) increases from 2 to 32. The chart (a), (b), (c), and (d) show the distances with the averaging intervals (I) of 2, 4, 8, and 16, respectively.

model to explore a similar size of parameter space regardless of the P setting.

Figure 2 presents the distances between local models and the averaged model in ResNet-20 training on CIFAR10. The local batch size B is set to 128 and the learning rate is fixed to 0.1. The model averaging interval I is set to 2, 4, 8, and 16 for the chart (a), (b), (c), and (d), respectively. When $P \leq 4$, we commonly see that the distance effectively increases as P increases in all the four charts. When P increases beyond 4, the distance is not much increased or even reduced. Regardless of the model averaging interval, therefore, we can expect the model to move more slowly on the parameter space when using more than 4 workers.

A slow movement of model can cause two potential problems as follows. First, it is equivalent to using a smaller learning rate. It has already been known that the learning rate should be sufficiently large to avoid falling into local minima that poorly generalize to the test data [2], [5], [15]. Second, given the same number of epochs, the model explores a smaller region of the parameter space. Some researchers have shown that a certain degree of noise in gradients makes the model explore more parameter space which results in having a better chance to achieve a high model accuracy [16], [17].

By comparing the validation accuracy among different P settings, we can indirectly view the impact of the model averaging on the quality of the training output. The model averaging interval is fixed to 8. The validation accuracy achieved after 200 epochs with $P = 2$ and $P = 4$ is almost the same ($91.98 \pm 0.2\%$ and $91.41 \pm 0.2\%$). When $P = 8$ and $P = 16$, we observed that the accuracy is significantly dropped ($86.08 \pm 0.1\%$ and $83.49 \pm 0.2\%$). The similar experimental results have been reported in [1]. These results demonstrate that a large P makes the model moves more slowly and it results in degrading the generalization performance.

Note that these results can be also analyzed using the concept of 'effective batch size' of local SGD. Given a fixed local batch size B, local SGD handles P mini-batches at each iteration. So, we can consider the 'effective batch size' is BP. Lin et al [1] empirically studied the impact of the effective batch size on the validation accuracy. Their key observation is that the larger the effective batch size, the lower the validation accuracy after the same number of epochs. They proposed *post-local SGD* that sets P to 1 until the learning rate is decayed. *Post-local SGD* takes advantage of a fast loss reduction at the early epochs while significantly sacrificing the degree of parallelism.

Based on our analysis, we consider the distance from local models to the averaged model as a practical metric for measuring how effectively the local models contribute to the training. We formally define the distance metric D as follows.

$$D = \frac{1}{P} \sum_{i=1}^{P} \|W^i - \bar{W}\|_2, \tag{9}$$

where W^i is the local model parameters of worker i and $\bar{W}$ is the model parameters averaged across all the workers. Based on our heuristics, we propose to tune the number of workers P as follows.

- *In order to scale up the local SGD training without a significant accuracy loss, increase P up to the threshold in which D stops increasing proportionally.*

Another advantage of this metric is that it enables to find the proper P without running the full training until the loss converges. As shown in Figure 2, the D values are flattened in the early few iterations regardless of P, B, and I settings. So, users can determine the best number of workers by running a few iterations with different P settings and comparing the distances. Considering the ever-increasing available training data and deep networks, this distance metric can dramatically reduce the time and resource for the hyper-parameter tuning.

Finding the model averaging interval – By increasing the model averaging interval I, the model parameters are synchronized less frequently and the overall communication cost is reduced. However, if I is too large, the local models can move towards different minima and it can result in making the model converge slowly. The statistical analysis presented in [2] also shows that a smaller I gives a better convergence rate. Therefore, I should be carefully tuned to make a good balance between the scalability and the convergence rate.

The recent works focus on how to maximize the interval without adversely affecting the convergence rate. Some re-

searchers have proposed adaptive model averaging methods [2], [5]. These methods adaptively adjust the model averaging interval based on the statistical analysis of the impact of I on the convergence rate at run-time.

In our proposed model averaging algorithm, the communication cost of the model averaging is shown in Equation 7. In the second term, the message size is $\frac{2M}{K}$, which means the communication cost is proportionally reduced as more processes are employed for each local model training. Note that the first term of Equation 3, the startup cost of each communication, is negligible in parallel neural network training due to the large message size w. Thanks to the reduce communication cost complexity, we found that the communication cost for model averaging is not significant compared to the cost of the computations and the gradient communications within each group. Therefore, when using our model averaging algorithm, we choose a sufficiently small I which gives a good convergence rate. In many previous works, I is usually set to a certain value between $10 \sim 128$. In our experiments, we found that, even when I is set to $2 \sim 4$, the communication cost for averaging the model parameters is almost negligible.

Adjusting the learning rate for local SGD – Like any other SGD-based algorithms, local SGD also requires users to properly adjust the learning rate to achieve a good model accuracy. Especially when P is large, as shown in Figure 2, the averaged model moves slowly and it makes the model easily fall into sharp minima. We present our learning rate adjustment method for local SGD as follows.

- Increase the learning rate as the number of workers increases.
- Apply gradual warm-up technique to each local model training.

First, we propose to increase the learning rate as the number of workers increases. It has been both empirically observed [1] and statistically analyzed [2], [3] in the previous works that increasing the number of workers yields a similar impact on the model accuracy as using a larger mini-batch size. As the batch size increases, the variance of the gradients goes down, and thus the training more easily converges to a sharp minimum that poorly generalizes to the test data [15]. Researchers have found that increasing the learning rate can alleviate such an effect [15], [18], [19]. Based on the same principle, if the learning rate is increased in local SGD training, the inherent noise scale can be considered to be increased. Smith et. al also proposed a concept of noise scale that is proportional to the learning rate in [20]. Thus, by averaging such noisy models in local SGD, the training does not easily converge into a minimum and keeps exploring the parameter space heading towards minima that better generalize to the test data (called 'flat' minima [15], [20]).

Second, we empirically found that the warm-up method helps stabilize the training when using larger learning rates. We employ the gradual warm-up method proposed in [18] within each local model such that the learning rate is initially set to μ_0 that is the best-tuned learning rate for the local batch size and then gradually increase it in the first few epochs. There are two known learning rate scaling methods for large batch synchronous SGD training: 'linear scaling rule' [18] and 'root scaling rule' [19]. Since the upper bound of learning rate that guarantees the convergence is problem-dependent, users should empirically find the maximum allowed learning rate considering the dataset size and other hyper-parameter settings. When the linear scaling rule is applied, the gradual warm-up increases the learning rate to $P\mu_0$. For the root scaling rule, the learning rate is increased to $\sqrt{P}\mu_0$.

IV. Performance Evaluation

We evaluate the proposed local-SGD training algorithm for deep neural networks. We apply our parallel training strategy to two representative classification and regression problems, CIFAR10 classification with ResNet-20 and DIV2K image super-resolution with EDSR.

Systems – All our experiments are carried out on Cori, a Cray XC40 supercomputer at National Energy Research Scientific Computing Center (NERSC). Each compute node has an Intel Xeon Phi Processor 7250, Knights Landing (KNL), that has 68 cores. AVX-512 vector pipelines with a hardware vector length of 512 bits are available at each node. The system has Cray Aries high-speed interconnections with 'dragonfly' topology.

Datasets – CIFAR10 [21] is a popular benchmark dataset for deep learning study. The dataset consists of 50,000 training samples and 10,000 validation samples. Each sample is an 3-channel image of size 32×32. DIV2K is a dataset from NTIRE2017 Super-Resolution Challenge [22], which contains 800 high-quality 2K resolution pictures. We used bicubic low-resolution images as the training data.

Models – For CIFAR10 classification, we used ResNet-20 [23] which is also one of the most popular CNN model architecture. The model has 20 layers in total, including convolution layers, pooling layers, and fully-connected layers. The layers also have residual connections between non-adjacent layers. For DIV2K image regression, we used EDSR [24] which is a deep CNN model. The model has 32 residual blocks each of which has 2 convolution layers and a residual connection between the input and the output of the block.

Software – We developed our own deep learning framework that is designed for parallel training on distributed-memory platforms. The software framework adopts MPI-OpenMP programming model. Each MPI process uses OpenMP to utilize all the cores for kernel functions such as matrix operations. Each process also employs a communication-dedicated POSIX thread that performs blocking MPI communications. In this way, the computation and communication can be explicitly overlapped based on users' overlapping strategy.

We chose to fix the number of training epochs and compare the model accuracy. It has been empirically shown that a large batch training can achieve a similar accuracy to the training with the best-tuned small batch size if the model is trained for proportionally increased epochs [19]. However, in practice, researchers usually stop the training when an

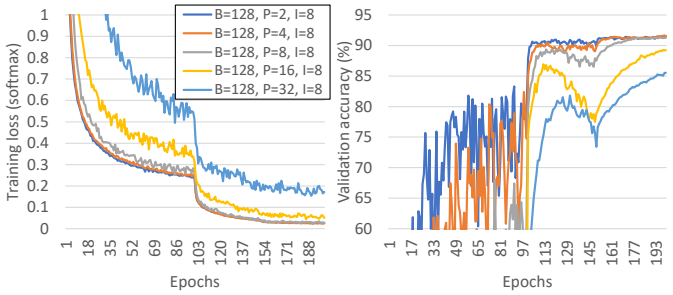

Fig. 3. Comparison of learning curves for CIFAR10 classification with ResNet-20 across different numbers of workers P. We fixed the local batch size B to 128 and the model averaging interval I to 8. Both the training loss and the validation accuracy are degraded when $P \geq 8$.

TABLE I
VALIDATION ACCURACY OF CIFAR10 CLASSIFICATION WITH RESNET-20. THE LOCAL BATCH SIZE B IS 128, THE MODEL AVERAGING INTERVAL I IS 8. THE LEARNING RATE μ FOR $P = 1$ IS 0.1 AND IT PROPORTIONALLY INCREASES AS P INCREASES.

B	P	I	μ	accuracy (%)
128	1	1	0.1	91.91%
128	2	8	0.2	91.61%
128	4	8	0.4	91.58%
128	8	8	0.8	91.11%
128	16	8	1.6	89.28%
128	32	8	3.2	85.55%

acceptable accuracy is achieved, rather than waiting until the training loss entirely converges. Considering such a common practice, we do not focus on adjusting the training epochs to achieve a better accuracy.

A. Comparison with Local SGD Variants

There are several algorithms that seek to improve upon local SGD, including *post-local* SGD [1], Sparse Aggregation SGD (SASGD) [4], and local SGD with adaptive model averaging interval [2], [5]. We found that all these algorithms use *allreduce* operations to average the model parameters across all the workers. *post-local* SGD begins with synchronous SGD and then increases the number of local models after decaying the learning rate. Once the number of local models increases, the model parameters are averaged using the *allreduce* operations. SASGD also averages the accumulated local gradients using *allreduce* operations. The adaptive model interval adjustment methods aim to minimize the frequency of the communications, however the communication pattern for averaging the model parameters is still the same.

We compare scaling performance among three different parallel training methods: synchronous SGD, *allreduce*-based local SGD, and our proposed local SGD. Although all the local SGD variants have different hyper-parameter adjustment methods or parameter update rule, they share the same communication pattern for averaging the model parameters. Therefore, we categorize all such algorithms to '*allreduce*-based local SGD'. Note that our proposed model averaging algorithm can be independently applied to any local SGD variants because it can directly replace the *allreduce* operations.

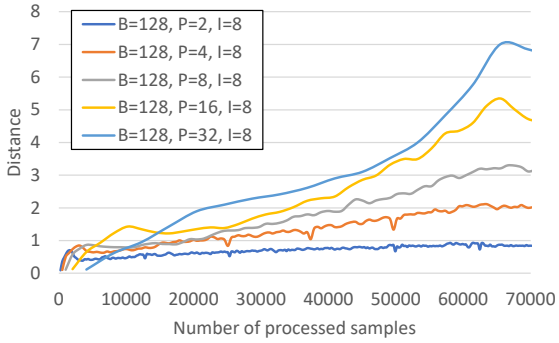

Fig. 4. Comparison of average distance between the local models and the averaged model. B is set to 128, I is set to 8. P varies between 2 and 32. The learning rate is initially set to 0.1 and gradually increased to 0.1P in the first five epochs (gradual warm-up). Compared to Figure 2, the distance is more effectively increased as P increases.

B. CIFAR10 Classification

Classification performance – Figure 3 shows the training loss curves (left) and the validation accuracy curves (right) with different P settings. Table I presents the best validation accuracy in the last 10 epochs. We report the accuracy results that are averaged across three separate experiments. The local batch size B is fixed to 128 and the number of workers varies between 1 and 32. The model averaging interval I is set to 8. For synchronous SGD ($P = 1$), we use a learning rate of 0.1 that is the best-tuned value for $B = 128$. We proportionally increase the learning rate for local SGD as P increases. The gradual warm-up is applied to the first five epochs when $P \geq 2$. We can observe that a comparable validation accuracy to the synchronous SGD with the best-tune batch size is achieved only when $P \leq 4$. Note that we consider that the accuracy drop is significant if it is larger than 0.5%

We demonstrate the effectiveness of the proposed distance metric D by comparing the learning curves and the distance curves. Figure 4 presents the distance curves measured from the CIFAR10 classification experiments. When $P = 2$, the distance is rapidly flattened and becomes around $0.8 \sim 0.9$. When P increases to 4, the distance is also proportionally increased to $1.9 \sim 2.0$. As shown in Table I, the difference of validation accuracy between $P = 2$ and $P = 4$ is negligible. As P further increases to 8, the distance is increased to $3 \sim 3.4$ that is slightly lower than the expected one (~ 3.9). Likewise, the validation accuracy is dropped by $\sim 0.4\%$. We can clearly see that D and P are no longer proportional to each other when $P > 8$. As shown in Figure 3, when $P > 8$, the training loss converges much slowly and the validation accuracy is also dropped significantly. These results empirically prove that the distance between the local models and the averaged model indirectly represents how effectively the workers contribute to minimizing the shared cost function in local SGD. Note that Figure 4 shows much longer distances than shown in Figure 2. The difference comes from the increased learning rate based on the learning rate control proposed in Section III-C.

Scaling performance – We found that the maximum P

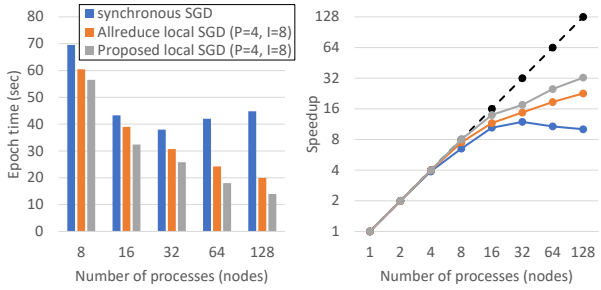

Fig. 5. Comparison of the epoch time (left) and speedup (right) for CIFAR10 classification among synchronous SGD, *allreduce*-based local SGD, and the proposed local SGD. For synchronous SGD, we used the batch size of 512. For local SGD, the local batch size B is 128, the number of workers P is 4, and the model averaging interval I is 8.

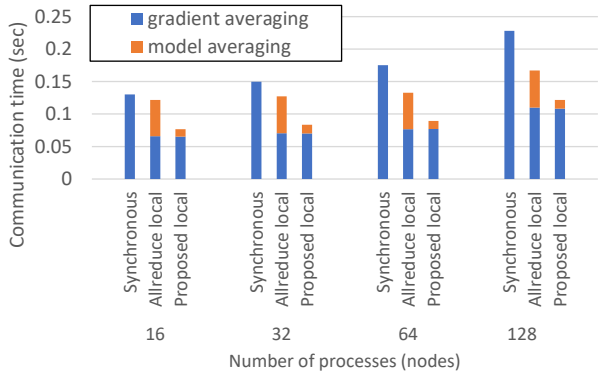

Fig. 6. Comparison of the communication time per iteration for ResNet-20 training among synchronous SGD, *allreduce*-based local SGD, and the proposed local SGD.

providing a good accuracy in local SGD training for CIFAR10 classification is 4, using our proposed distance metric. So, we study the scaling performance using 4 workers. That is, when scaling up, the number of local models is fixed to 4 and the number of processes increases for data parallel training of each local model. We compare the performance among synchronous SGD, *allreduce*-based local SGD, and the proposed local SGD.

Figure 5 shows the average epoch time (left) and speedup (right). The local batch size B is 128 and the model averaging interval I is 8. Note that when the number of processes is smaller than P, we performed synchronous SGD. For synchronous SGD, to fairly compare the scaling performance, we used a large batch size of 512. The speedup of synchronous SGD is flattened when the training is scaled up to 32 processes. In data parallel training, as the number of processes increases, the computational workload is proportionally reduced while the communication cost increases. Therefore, the communication time ends up being dominant over the computation time and it results in flattening the speedup.

We can see that the proposed local SGD significantly outperforms synchronous SGD. It achieves almost a linear speedup until 16 processes and then the scaling efficiency is degraded as more processes are employed. *allreduce*-based approach also shows better scaling performance than synchronous SGD, however, the speedup is flattened from 64 processes. Because the proposed model averaging algorithm does not affect the computational workload, the timing difference between *allreduce*-based approach and the proposed approach comes only from the reduced communication cost.

Figure 6 presents the communication time comparison among the three parallel training methods. We measured the gradient averaging communications and the model averaging communications separately. First, local SGD training has a shorter communication time than synchronous SGD. Because the gradients are averaged within each group only, the communication cost is reduced as more workers are employed. Second, our proposed model averaging algorithm significantly reduces the model averaging communication cost compared to the *allreduce*-based approach. This communication time comparison clearly shows how effectively our proposed algorithm improves the scalability of local SGD.

It is worth noting that the communication cost of model averaging is a constant regardless of the overall number of processes. Since the model is averaged across all the groups, the communication cost is affected only by the number of workers P. When scaling up the training, we fix the number of workers P to 4 and increase the group size K. So, the communication cost of gradient averaging increases as the number of processes increases, while the communication cost of model averaging stays the same.

C. DIV2K Super-Resolution

Image super-resolution is one of the popular regression problems. We use Enhanced Deep Super-Resolution (EDSR) [24], a residual network that has 32 residual blocks each of which has two convolution layers. This model is much deeper and larger than ResNet-20 used in the previous experiment.

Regression performance – For DIV2K super-resolution experiments, we use Peak Signal-to-Noise Ratio (PSNR) as a similarity metric between the estimated high-resolution images and the original images. We follow all the hyper-parameter settings used in [24], except the batch size and learning rate. The model is trained using Adam [25], a variant of the classical SGD, for 1200 epochs. Note that, the EDSR training diverges when P increases to 8 and the learning rate is proportionally increased. So, in order to keep the learning rate from being increased out of the safe range, we multiply $\sqrt{P}$ to the learning rate as P increases, as shown in Table II.

Figure 7 shows the learning curves with different P settings. Table II shows the best validation accuracy in the last 100 epochs. We observe that almost the same validation accuracy is achieved when $P \leq 8$. When $P = 8$, the accuracy achieved after 1200 epochs is 33.89 dB which is just 0.36 dB lower than that of synchronous training. However, when $P = 16$, the validation accuracy is dramatically dropped to 30.95 dB.

Figure 8 presents the distance from the local models to the averaged model with different P settings. We can see the same pattern that has been shown in Figure 4. When $P = 2$, the distance D is measured to be $0.02 \sim 0.03$. As P increases, the distance curve also moves up proportionally until $P = 8$

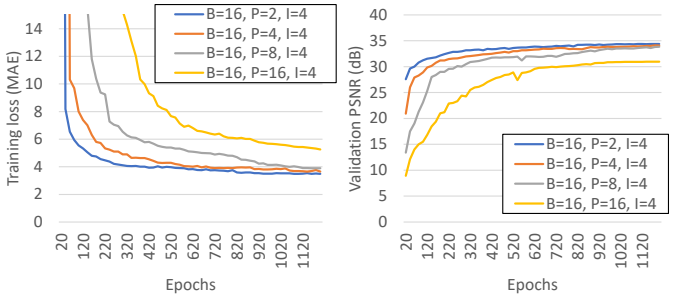

Fig. 7. Comparison of learning curves for DIV2K image super-resolution with EDSR across different numbers of workers P. We fixed the local batch size B to 16 and the model averaging interval I to 4. Both the training loss and the validation accuracy are degraded when $P \geq 16$.

TABLE II
VALIDATION ACCURACY OF DIV2K SUPER-RESOLUTION WITH EDSR. THE LOCAL BATCH SIZE B IS 16, THE MODEL AVERAGING INTERVAL I IS 4. THE LEARNING RATE μ FOR $P = 1$ IS 0.0001. AS P INCREASES, WE INCREASE THE LEARNING RATE BY MULTIPLYING A SQUARE ROOT OF THE INCREASED RATIO.

B	P	I	μ	accuracy (PSNR)
16	1	1	0.1	34.35 dB
16	2	4	0.1414	34.39 dB
16	4	4	0.2	34.10 dB
16	8	4	0.2828	33.89 dB
16	16	4	0.4	30.95 dB

($0.04 \sim 0.07$ when $P = 4$ and $0.07 \sim 0.13$ when $P = 8$). Thus, we consider that the workers still effectively contribute to adjusting the global model towards minima when $P \leq 8$. However, when $P = 16$, the distance is significantly reduced and slowly increases as the training progresses. This result is aligned with the accuracy drop shown in Table II.

Scaling performance – Based on our heuristics, given $B = 16$, we can find the maximum number of workers that provides a good accuracy to be 8 for DIV2K regression. So, we study the scaling performance using 8 workers. For synchronous training, we use the batch size of 128 for a fair comparison. Figure 9 shows the average epoch time (left) and the speedup (right). We see that the speedup of synchronous training is flattened as the number of processes is 64. Both local SGD approaches effectively scale up to 64 processes, and our proposed local SGD outperforms the *allreduce*-based approach. Figure 10 shows the communication time comparison. Similarly to Figure 6, both local SGD approaches have a cheaper gradient averaging cost than synchronous SGD. Additionally, our proposed model averaging algorithm significantly reduces the model averaging cost and it results in achieving a better speedup as shown in Figure 9.

D. Comparison with TensorFlow

TensorFlow with Horovod is one of the most popular software frameworks for deep learning. We compare the scaling performance between TensorFlow and our own software framework ('PCNN'). This comparison demonstrates that our scaling performance study is based on a reasonably good baseline. We used TensorFlow 2.2.0 and Horovod 0.19.0,

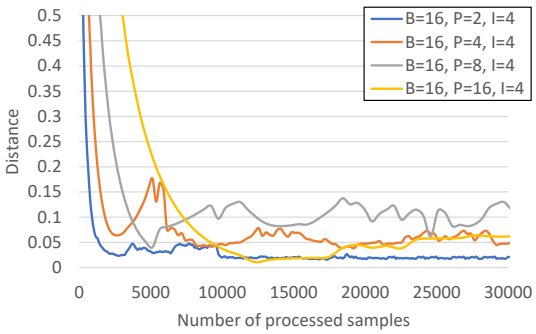

Fig. 8. Comparison of average distance between the local models and the averaged model. B is set to 16, I is set to 4. P varies between 2 and 16. The learning rate is initially set to 0.0001 and gradually increased to $0.1\sqrt{P}$ in the first five epochs (gradual warm-up).

the latest versions supported on Cori. Figure 11 presents the execution time and speedup of synchronous SGD training. We see PCNN outperforms TensorFlow when the number of processes is larger than or equal to 16. The performance difference largely comes from the communication overlap. Horovod performs *allreduce* communications after TensorFlow returns the gradients. Thus, the communications cannot be overlapped with any computations and the whole communication time is exposed degrading the scaling efficiency. PCNN adopts the overlapping strategy proposed in [10]. In Figure 5 and Figure 9, 'synchronous SGD' represents the synchronous parallel training performance of our framework.

V. CONCLUSION

In this paper, we discussed how to exploit the degree of parallelism in local SGD neural network training. We proposed a hierarchical parallelism that applies data parallelism to each local model training. When scaling up, we fix the number of workers to the best-tuned value and increase the number of processes for each local model training. The number of workers is tuned using the proposed distance metric that evaluates how effectively each local model contributes towards minimizing the shared cost function. Our experimental results demonstrate that the proposed hierarchical parallelism enables to achieve significantly improved scaling performance while maintaining the model accuracy. This paper can provide a guideline for large-scale deep learning applications to improve the scaling efficiency with a minimal extra hyper-parameter tuning. We believe that applying the existing large-batch training techniques to each local model training and studying the impact can be an interesting future work.

ACKNOWLEDGMENT

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Scientific Discovery through Advanced Computing (SciDAC) program, under the "RAPIDS Institute". This work is also supported in part by the DOE awards DE-SC0021399, DE-SC0014330, DE-SC0019358, and NIST award 70NANB19H005. This research used resources of

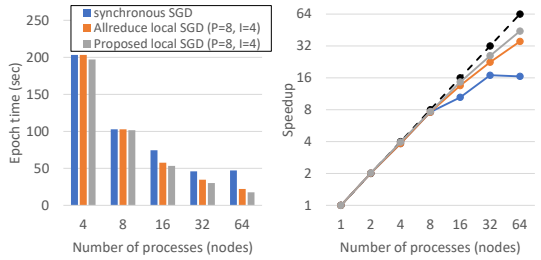

Fig. 9. Comparison of the epoch time (left) and speedup (right) for DIV2K regression among synchronous SGD, *allreduce*-based local SGD, and the proposed local SGD. For synchronous SGD, the batch size is set to 128. For local SGD, the local batch size B is 16, the number of workers P is 8, and the model averaging interval I is 4.

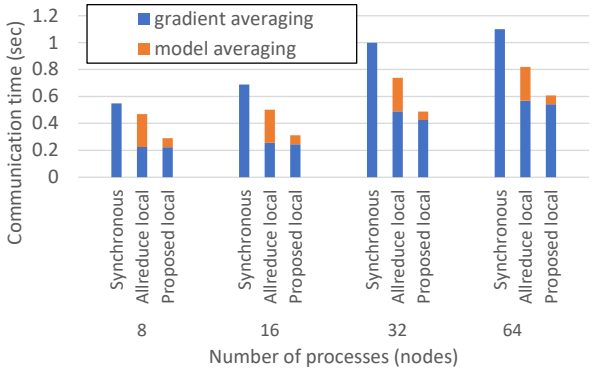

Fig. 10. Comparison of the communication time per iteration for EDSR training (synchronous SGD, *allreduce*-based local SGD, and the proposed local SGD).

the National Energy Research Scientific Computing Center, a DOE Office of Science User Facility supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

REFERENCES

[1] T. Lin, S. U. Stich, K. K. Patel, and M. Jaggi, "Don't use large mini-batches, use local sgd," *arXiv preprint arXiv:1808.07217*, 2018.

[2] F. Haddadpour, M. M. Kamani, M. Mahdavi, and V. Cadambe, "Local sgd with periodic averaging: Tighter analysis and adaptive synchronization," in *Advances in Neural Information Processing Systems*, 2019, pp. 11 080–11 092.

[3] H. Yu, S. Yang, and S. Zhu, "Parallel restarted sgd with faster convergence and less communication: Demystifying why model averaging works for deep learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 5693–5700.

[4] G. Cong, O. Bhardwaj, and M. Feng, "An efficient, distributed stochastic gradient descent algorithm for deep-learning applications," in *2017 46th International Conference on Parallel Processing (ICPP)*. IEEE, 2017, pp. 11–20.

[5] J. Wang and G. Joshi, "Adaptive communication strategies to achieve the best error-runtime trade-off in local-update sgd," *arXiv preprint arXiv:1810.08313*, 2018.

[6] S. Lee, D. Jha, A. Agrawal, A. Choudhary, and W.-k. Liao, "Parallel deep convolutional neural network training by exploiting the overlapping of computation and communication," in *2017 IEEE 24th International Conference on High Performance Computing (HiPC)*. IEEE, 2017, pp. 183–192.

[7] S. H. Hashemi, S. A. Jyothi, and R. H. Campbell, "Tictac: Accelerating distributed deep learning with communication scheduling," *arXiv preprint arXiv:1803.03288*, 2018.

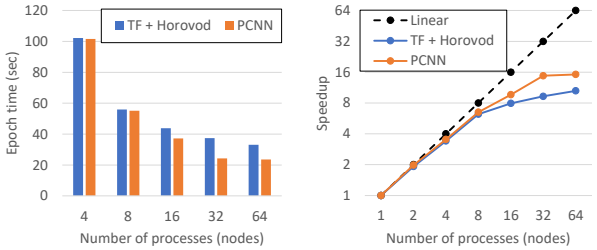

Fig. 11. Comparison of the scaling performance between TensorFlow + Horovod and our own software framework (PCNN). The training is scaled up using synchronous SGD. The mini-batch size is 128.

[8] S. Wang, A. Pi, and X. Zhou, "Scalable distributed dl training: Batching communication and computation," in *Proc. of AAAI*, 2019.

[9] H. Wang, S. Guo, and R. Li, "Osp: Overlapping computation and communication in parameter server for fast machine learning," in *Proceedings of the 48th International Conference on Parallel Processing*, 2019, pp. 1–10.

[10] S. Lee, A. Agrawal, P. Balaprakash, A. Choudhary, and W.-K. Liao, "Communication-efficient parallelization strategy for deep convolutional neural network training," in *2018 IEEE/ACM Machine Learning in HPC Environments (MLHPC)*. IEEE, 2018, pp. 47–56.

[11] M. Barnett, L. Shuler, R. van De Geijn, S. Gupta, D. G. Payne, and J. Watts, "Interprocessor collective communication library (intercom)," in *Proceedings of IEEE Scalable High Performance Computing Conference*. IEEE, 1994, pp. 357–364.

[12] R. Thakur, R. Rabenseifner, and W. Gropp, "Optimization of collective communication operations in mpich," *The International Journal of High Performance Computing Applications*, vol. 19, no. 1, pp. 49–66, 2005.

[13] R. W. Hockney, "The communication challenge for mpp: Intel paragon and meiko cs-2," *Parallel computing*, vol. 20, no. 3, pp. 389–398, 1994.

[14] R. Rabenseifner, "A new optimized mpi reduce and allreduce algorithm," 1997.

[15] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang, "On large-batch training for deep learning: Generalization gap and sharp minima," *arXiv preprint arXiv:1609.04836*, 2016.

[16] A. Neelakantan, L. Vilnis, Q. V. Le, I. Sutskever, L. Kaiser, K. Kurach, and J. Martens, "Adding gradient noise improves learning for very deep networks," *arXiv preprint arXiv:1511.06807*, 2015.

[17] M. Zhou, T. Liu, Y. Li, D. Lin, E. Zhou, and T. Zhao, "Towards understanding the importance of noise in training neural networks," *arXiv preprint arXiv:1909.03172*, 2019.

[18] P. Goyal, P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He, "Accurate, large minibatch sgd: Training imagenet in 1 hour," *arXiv preprint arXiv:1706.02677*, 2017.

[19] E. Hoffer, I. Hubara, and D. Soudry, "Train longer, generalize better: closing the generalization gap in large batch training of neural networks," in *Advances in Neural Information Processing Systems*, 2017, pp. 1731–1741.

[20] S. L. Smith, P.-J. Kindermans, C. Ying, and Q. V. Le, "Don't decay the learning rate, increase the batch size," *arXiv preprint arXiv:1711.00489*, 2017.

[21] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Citeseer, Tech. Rep., 2009.

[22] R. Timofte, E. Agustsson, L. Van Gool, M.-H. Yang, L. Zhang, B. Lim, S. Son, H. Kim, S. Nah, K. M. Lee *et al.*, "Ntire 2017 challenge on single image super-resolution: Methods and results," in *Computer Vision and Pattern Recognition Workshops (CVPRW), 2017 IEEE Conference on*. IEEE, 2017, pp. 1110–1121.

[23] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[24] B. Lim, S. Son, H. Kim, S. Nah, and K. Mu Lee, "Enhanced deep residual networks for single image super-resolution," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2017, pp. 136–144.

[25] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.