Improving Scalability of Parallel CNN Training by Adjusting Mini-Batch Size at Run-Time

Sunwoo Lee*, Qiao Kang*, Sandeep Madireddy†, Prasanna Balaprakash†, Ankit Agrawal*,
Alok Choudhary*, Richard Archibald‡, and Wei-keng Liao*

*Northwestern University
{slz839,qkt561,ankitag,choudhar,wkliao}@eecs.northwestern.edu
†Argonne National Laboratory
{smadireddy,pbalapra}@anl.gov
‡Oak Ridge National Laboratory
archibaldrk@ornl.gov

***Abstract*—Training Convolutional Neural Network (CNN) is a computationally intensive task, requiring efficient parallelization to shorten the execution time. Considering the ever-increasing size of available training data, the parallelization of CNN training becomes more important. Data-parallelism, a popular parallelization strategy that distributes the input data among compute processes, requires the mini-batch size to be sufficiently large to achieve a high degree of parallelism. However, training with large batch size is known to produce a low convergence accuracy. In image restoration problems, for example, the batch size is typically tuned to a small value between $16 \sim 64$, making it challenging to scale up the training. In this paper, we propose a parallel CNN training strategy that gradually increases the mini-batch size and learning rate at run-time. While improving the scalability, this strategy also maintains the accuracy close to that of the training with a fixed small batch size. We evaluate the performance of the proposed parallel CNN training algorithm with image regression and classification applications using various models and datasets.**

***Index Terms*—Deep Learning, Convolutional Neural Network, Parallelization, Adaptive Batch Size**

I. Introduction

Deep Convolutional Nerual Network (CNN) has been used in a variety of applications such as image classification [1], [2], restoration [3]–[6], object detection [7], [8], and super-resolution imaging [9]–[11]; While users enjoy its success, training deep neural networks is, in fact, an extremely computationally intensive task that can take hours or even days to complete. Considering the ever-increasing size of available training data, efficient parallelization is crucial to finish the training in a reasonable amount of time. The most popular CNN training algorithm is synchronous Stochastic Gradient Descent (SGD) [12]. The algorithm iteratively calculates gradients of a cost function with respect to the network parameters from a random subset of training samples (mini-batch). Then, the parameters are updated using the averaged gradients. In data-parallel synchronous SGD, each mini-batch is evenly distributed to all workers and concurrently processed. This parallelization strategy exhibits a strong data dependency between any two consecutive iterations, i.e. iteration $(i+1)$ cannot proceed before the completion of iteration i. Without the possible cross-iteration concurrency, the degree of parallelism is limited by the number of data samples in a mini-batch. Thus, increasing mini-batch size becomes an intuitive approach to employ more workers in the hope of reducing execution time.

Several recent parallelization works presented performance results scalable up to thousands of nodes using large mini-batch sizes [13]–[16]. However, most of them also acknowledged that using large batch sizes can result in achieving a lower validation accuracy. The impact of batch sizes on the accuracy has been statistically analyzed in [17]–[19]. Figure 1 illustrates such impact using an example of an EDSR [11] training on the DIV2K super-resolution image dataset [20]. Each learning curve corresponds to mini-batch sizes, ranging from 16 to 256 images. When the batch size increases, the training converges more slowly (in the number of epochs) and achieves a lower validation accuracy. Such similar trends of learning curves are also shown in [14], [18], [19], [21]–[24]. Owing to this observation, we argue two evaluation principles below in order to ensure a fair performance comparison among different neural network training methods.

- Timing comparison is only fair among methods that produce the same model accuracies or within a small, tolerable margin.
- The training time should be measured from the beginning until the accuracy converges to a stable value.

The former argues a fair comparison under the condition of the same input and output. As shown in Figure 1, $B = 128$ and $B = 256$ give much lower accuracy than smaller B values. Models produced with less accuracy are usually regarded of no use to domain scientists. The latter describes the unique characteristics of neural networks whose training process is not considered completed until the convergence condition is met. This argument stems from our study of recent parallelization works that measured the time up to a fixed number of epochs to represent the performance of a training method when calculating the speedups and comparing against other methods. In this paper, we present our experimental results and analysis by following the above two principles.

978-1-7281-0858-2/19/$31.00 ©2019 IEEE

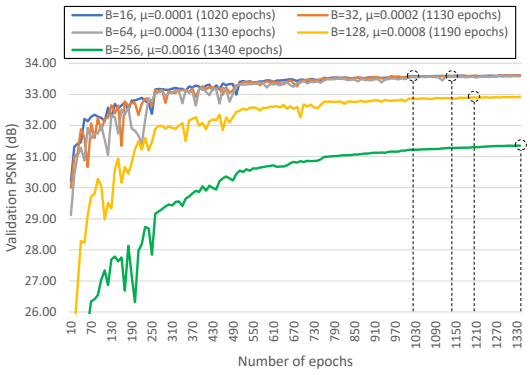

Fig. 1. Learning curves for EDSR training on DIV2K dataset. B is the mini-batch size, μ is the learning rate, and the numbers shown in brackets are the number of epochs till model converged. The training terminates when the validation accuracy has not increased for 50 consecutive epochs. Batch sizes larger than 64 result in significantly lower accuracy.

We propose a parallel CNN training strategy that adjusts the mini-batch size during the training. The common practice of neural network training is to tune the mini-batch size to a small value that produces the best accuracy. Especially for image restoration or super-resolution problems, the mini-batch size is typically tuned to a small value between $16 \sim 64$ [3]–[6], [9]–[11], which is too small to effectively scale up the parallel training. Our goal is to improve the degree of parallelism without a significant loss in validation accuracy. In our design, the training begins with small batch size and it gradually increases. To increase the batch size without affecting the gradient noise scale, we also adjust the learning rate as the batch size increases. The interval of batch size increase is adaptively determined based on the ratio of cost reduction to the distance between the initial parameters and the latest ones. We also propose to dramatically lower the learning rate when the training cost is saturated, to keep the generalization performance from being degraded.

Besides adjusting the mini-batch size and learning rate, our parallelization strategy also focuses on the overlapping of communication and computation. In data-parallel trainings, the locally computed gradients are averaged among all workers before updating the parameters. We implement the averaging with MPI all-to-all personalized communication followed by a local summation and an MPI allgather communication. By having two separate communications, not only the backpropagation but also the feed-forward computations in the next iteration can overlap the communications.

We evaluate the performance of the proposed methods with image regression and classification tasks on the KNL nodes of Cori supercomputer at NERSC. For image regression experiments, we train Enhanced Deep Super-Resolution (EDSR) [11] on DIV2K [20] and Phantom [25] datasets and compare the performance among various training methods. For image classification experiments, we train ResNet20 [1] on Cifar10 which is one of the most popular benchmark dataset. The DIV2K training with the best-tuned batch size 16 scales up to 16 KNL nodes (1,088 cores) only achieving 9.54 speedup. The Phantom training with the best-tuned batch size 16 achieves a speedup of 5.64. Our proposed training strategy increases the batch size up to 256 during the training and the parallel training achieves a speedup of 81.71 using up to 256 KNL nodes (17,408 cores) while maintaining the similar validation accuracy to that of the training with the best-tuned batch size. The Cifar10 training with the best-tuned batch size 128 achieves a speedup of 3.49 only due to the high ratio of communication to computation. Our proposed training strategy successfully increases the batch size up to 2048 and the training is scaled up to 256 nodes achieving a speedup of 64.64. The classification accuracy difference is lower than 1%.

II. Background

A. Convolutional Neural Network

CNN is a type of artificial neural network that contains convolution layers [26]. The convolution layers have a connection pattern such that each neuron is connected to a subset of neurons at the previous layer. Such connection pattern enables exploitation of spatially-local correlation in the input data. Each convolution layer can be followed by a pooling layer depending on the model architecture.

Recently, residual network has been proposed which has connections between non-adjacent layers [1]. The non-adjacent connection is called 'residual connection'. The residual connection makes each layer compute the output using data from not only the neighboring layer but also the non-adjacent layers so that it can effectively tackle the problem of vanishing gradient [27]. The residual networks have been popularly used for a variety of applications [1], [2], [9]–[11], [14], [16].

B. Mini-Batch Training

The batching method has been widely used for neural network training. The batching algorithm stochastically approximates the gradients by computing them from a random subset of training samples instead of the entire dataset. This approach has shown superior results than other approaches and is widely used for neural network trainings [27].

The batching approach causes a data dependency of the model parameters across iterations. Each mini-batch can be processed only after the parameters are updated using the previous mini-batch. Therefore, the maximum workload that can be parallelized is a single iteration for processing a mini-batch. Unfortunately, the common practice of training is to fix the batch size to a small value which gives the best convergence rate. For example, the batch size for image classification is usually fixed to $128 \sim 256$ during the training [1], [21], [28]. The image regression applications also use a fixed small batch size between $16 \sim 64$ [3]–[6], [9]–[11]. Such small batch sizes make it challenging to scale up the training.

III. CNN Training with Adaptive Batch Size and Learning Rate

In this section, we discuss the problems in large batch training and our solutions to them. We begin with describing the impact of the large batch size on the training result as

well as the potential problems. Then, we present our training strategy which addresses the described problems by adaptively adjusting the batch size and learning rate at run-time.

A. Impact of Batch Size on Model Accuracy

In this paper, we consider minimization problems of the form

$$F(w) = \frac{1}{N}\sum_{i=1}^{N} f(w, x_i), \tag{1}$$

where N is the number of training samples, w is the model parameters, x_i is the i^{th} training sample, and f is the cost function of w and x.

Mini-batch SGD computes the gradients from a random subset of training samples (which is called mini-batch). The stochastic gradients can be considered as a random variable with mean of $\nabla F(w)$. Based on Central Limit Theorem, the variance of the random variable is inversely proportional to the mini-batch size [29]. If the variance is large, the gradients can be considered as noisy. Smith et al. analyzed the impact of the batch size on the gradient noise scale [17]. The noise scale describes the correlation between the batch size and the random fluctuation of SGD dynamics. For SGD, the noise scale is approximated by the following equation under the assumption of $N \gg B$.

$$g \approx \mu\frac{N}{B} \tag{2}$$

This analysis shows that, when using a large batch size, the learning rate should be proportionally increased to have the same noise scale. Their experimental results demonstrate that the analysis can be applied to variants of SGD such as momentum SGD and Adam [30]. Goyal et al. proposed 'linear scaling rule' in [21] that can be explained by this analysis. The authors empirically showed that large batch sizes can be used for ImageNet classification when the learning rate is proportionally increased. Hoffer et al. have proposed 'root scaling rule' in [31]. The authors argue that the variance of the stochastic gradients is proportional to $\frac{\mu^2}{B}$. Their statistical analysis is that the variance can stay the same when the learning rate is proportional to the square root of the batch size as follows.

$$\mu \propto \sqrt{B} \tag{3}$$

Recall that the stochastic gradient is considered as a random variable with mean of $\nabla F(w)$. By making the variance stay the same, one can expect the similar convergence rate.

Although these two works have derived different update rules, they provide a common insight that the learning rate should be increased when using a large batch size. In practice, the gradients should be sufficiently noisy to achieve a good accuracy [17], [18], [29]. Especially, in non-convex problems such as neural network trainings, the noisy gradients help the model avoid falling into a sharp minima which poorly generalizes to the test dataset.

B. Impact of Batch Size on Parallel Performance

When using the data parallelism strategy, the degree of parallelism depends on the problem size that can be partitioned among all the available processes. In the case of CNN training, it is the mini-batch size, as the minimum indivisible unit of workload that can be assigned to a process is a single training sample. Data parallelism partitions the samples in each mini-batch evenly among all the processes. The highest degree of parallelism is thus B. Therefore, larger mini-batch size enables a parallel algorithm to run on more processes.

Using a large batch size can also reduce the communication cost per epoch. Recall the communication for each batch is to average the gradients across all the processes. Given a network, the number of model parameters is independent from the batch size. In other words, the communication amount for averaging the gradients is not affected by the batch size. However, the number of communications is equal to the number of mini-batches in each epoch, N/B. Increasing the value B effectively reduces the value of N/B.

In terms of computation, the amount of parameter updates per epoch iteration is also reduced when using a large batch size. Given N training samples, the number of parameter updates per epoch is $\frac{N}{B}$, where B is the batch size. Since each parameter update takes the same amount of computation, increasing B proportionally reduces the number of updates.

C. Problems in Training with Large Batch Size

In this paper, we focus on two problems that can be observed in large batch trainings.

- The training cost $F(w)$ is not effectively reduced yielding a poor convergence accuracy.
- The model easily loses generalization performance.

First, the large batch size causes a low variance of the stochastic gradients and SGD quickly converges providing a low convergence accuracy. This problem can be alleviated with warm-up techniques such that the training starts with a small learning rate and then increases it after a pre-defined number of epochs. However, if the batch size is larger than a certain problem-dependent threshold, the cost is still not effectively reduced [21], [22]. Second, it is already known that large batch sizes can make the model lose the generalization performance [18], [23]. In other words, the large batch training tends to over-fit the model so that the cost function is well minimized while providing a low validation accuracy.

We address these two problems by adjusting hyper-parameters during training. In the following subsections, we discuss how to address the described problems with adaptive batch size and learning rate control methods.

D. Adaptive Batch Size Control

The main idea of our training strategy is to begin the training with a small batch size B_s and gradually increase the batch size during the training. As the batch size increases, we also increase the learning rate at run-time to minimize the impact of the increased batch size on the gradient noise scale. The

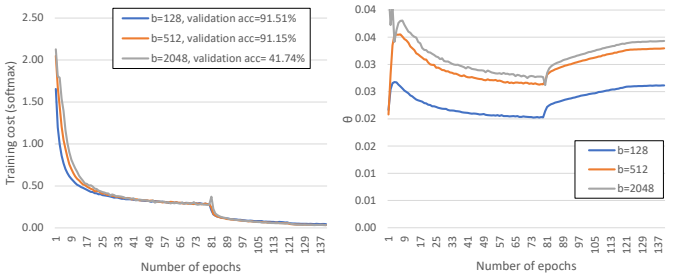

Fig. 2. The training cost curves (left) and θ curves (right) for ResNet20 training on Cifar10 datasets. All three batch sizes achieve almost the same training cost after 140 epochs. However, higher the θ curve, lower the validation accuracy. This result demonstrates that θ roughly shows how sharp the minimizer is. Note that the high θ at the beginning of 'b=2048' curve is due to the learning rate warmup.

batch size and learning rate are adjusted after every K epochs until the batch size reaches the maximum size B_m.

The small batch size at the early training epochs helps rapidly lower the training cost. So, K should be sufficiently large to effectively minimize the cost function. On the other hand, a large K indicates that the degree of parallelism is limited for more epochs due to the slow batch size increase. Our training algorithm aims to find the smallest K that effectively reduces the training cost.

We define a practical metric θ for estimating the sharpness of the minimizer.

$$\theta_i = \frac{F(w_0) - F(w_i)}{\|w_0 - w_i\|}, \tag{4}$$

where w_0 is the initial parameters. This metric shows the ratio of the cost reduction to the distance between the initial parameters and the latest ones. We can roughly estimate how sharp the current minimizer is by checking θ. Given a training cost $F(w_i)$, lower θ means the parameters have more moved to achieve the same cost reduction. Figure 2 shows the training cost curves (left) and the θ curves (right) of ResNet-20 training on Cifar10 dataset with varying batch sizes. Even though all the batch sizes achieve almost the same training cost after 140 epochs, the validation accuracy varies significatnly. We clearly see that higher the θ curve, lower the validation accuracy. This result demonstrates that θ can be considered as an indirect metric for measuring the sharpness of the minimizer.

We increase the batch size after the θ curve peaks so that the cost is sufficiently reduced before the batch size starts to increase. For example, in Figure 2, the batch size of 128 shows the peak θ at 5^{th} epoch. So, we set $K = 5$ so that the batch size increases after every 5 epochs. We also see that the θ curve for a large batch size peaks later than that of the smaller batch sizes. So, by increasing the batch size gradually after every K epochs, we can expect the batch size increases after the θ curve of each batch size has already peaked. Such careful adjustments also help avoid the cost fluctuation caused by the increased learning rate.

Algorithm 1 is a CNN training algorithm with the proposed adaptive batch size method. The algorithm iteratively traverses

Algorithm 1 SGD with Increasing Batch Size. (E: the number of epochs, N: the number of training samples, w_0: initial model parameters, μ_0: initial learning rate, B_s: the starting batch size, B_m: the maximum batch size, f: the cost function)

1: $w \leftarrow w_0, B \leftarrow B_s, \mu \leftarrow \mu_0, n \leftarrow 1, K \leftarrow \infty$
2: **while** stop condition is not met **do**
3: **for** $i \leftarrow 1 \cdots \frac{N}{B}$ **do**
4: $\mathcal{B} \leftarrow i^{th}$ mini-batch of size B.
5: $\nabla w \leftarrow$ Compute_Gradient(f, $\mathcal{B}$, w).
6: Update w using ∇w.
7: **if** K is ∞ **then**
8: Compute θ using Eq. 4.
9: **if** θ is not changed more than 10%, **then**
10: $K \leftarrow n$.
11: **if** $(n \bmod K) = 0$ **and** $B < B_m$ **then**
12: Increase both batch size B and learning rate μ
13: Increment n by 1

over all the training samples until the stop condition is satisfied. Typically, the training stops when either the parameters are not further adjusted due to the small gradients or a target accuracy is achieved. For each mini-batch $\mathcal{B}$, the gradients of a cost function f are computed with respect to the parameters at line 5. The parameters are updated using the averaged gradients at line 6. The θ is monitored and K is determined when θ starts to be saturated at line $7 \sim 10$. In this study, we consider θ is saturated if it is changed less than 10%.

When increasing the batch size and learning rate at line 12, based on the statistical analysis in [17], we adjust the batch size and learning rate together with a same factor to make the gradient noise scale stay the same. To force the convergence of SGD, we lower the gradient noise scale by decaying the learning rate once the training cost is saturated.

The batch size can be increased to a certain problem-dependent threshold without affecting the accuracy [18], [21]. For example, it has been shown that, the batch size for ImageNet classification can be increased to $4096 \sim 8192$ without affecting the accuracy [21]. We call this batch size 'maximum stable batch size'. By setting B_s to the maximum stable batch size, we can significantly improve the degree of parallelism. In practice, the maximum stable batch size can be easily found by comparing l2-norm of the gradients among batch sizes. A sufficient condition for gradients to be a descent direction with respect to the parameters is as follows [29].

$$\|\nabla w - \nabla F(w)\| < \|\nabla w\|$$

Note that the expected value of the left-hand side of the above inequality is the variance of the gradients [18]. Assuming the above condition is satisfied at most of the iterations, if two batch sizes give a similar l2-norm of the averaged gradients, it implies they have a similar maximum allowed variance of the gradients and they likely have a similar convergence property. In the later discussion, we assume B_s is set to the maximum stable batch size found by the described method.

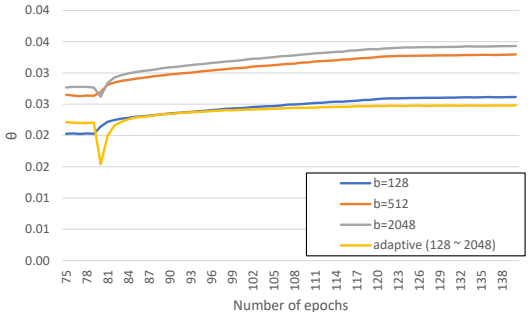

Fig. 3. The ratio of cost reduction to the distance between the initial parameters and the current ones, θ, for ResNet20 training on Cifar10 datasets. The proposed adaptive learning rate method keeps θ curve from being increased after the first learning rate decay step at 80^{th} epoch.

E. Adaptive Learning Rate Control

In Algorithm 1, the batch size increases to B_m most likely before the training loss is saturated. If the cost is minimized using such a large batch size, the model is easily attracted to a sharp minimizer. The generalization problem of large batch training has been already observed in many previous works [18], [23], [24].

To alleviate such effect, we intentionally lower the cost reduction speed after the first training cost saturation by dramatically lowering the learning rate. Once the training cost is saturated, one can decay the learning rate to further lower the cost. The decayed learning rate enables to fine-tune the parameters rather than exploring the parameter space, so that it further lowers the training cost and ends up converging into a minima. We consider step-wise learning rate decay with a decay factor β. Once the training cost is saturated, we scale down the learning rate to the initial learning rate μ_0 first and then multiply it by β. So, the effective learning rate decay factor becomes $\frac{B_s}{B_m}\beta$ for the first decay step. For the later decay steps, we use the fixed decay factor β only.

When the learning rate decays, as shown in Equation 2, we can expect a faster convergence rate due to the lower noise scale of the gradients. In our training algorithm, since the batch size has been increased to B_m, the variance of the gradients is lower than that of the small batch training. So, the model will rapidly converge to a minima which has a poor training cost. However, we found that such fast convergence enables to maintain the generalization performance even using the large batch size. Figure 3 shows the θ curves for ResNet-20 training on Cifar10 dataset after the first saturation of training cost. We see that θ of our proposed method does not significantly increase after the first learning rate decay step, while the other curves commonly increase. Note that our proposed method achieves 90.78% validation accuracy using up to 2048 batch size while the traditional SGD with the same batch size achieves 46.29% after 140 epochs.

IV. Parallel Training with Adaptive Batch Size

Our design is based on the data parallelism that distributes data among processes while keeping the model parameters duplicated. Since the minimum, indivisible data unit that can be assigned to each process is a training sample, for instance an image, we distribute the samples in each mini-batch evenly among all the processes. In this case, the maximum number of processes that can participate in the training is bounded by the number of samples in each mini-batch.

A. Data Parallelism with Increasing Batch Size

Our proposed training algorithm increases the batch size during the training. There are two possible design choices of parallelization. The first is to employ the number of processes equal to B_s, the starting batch size, so that the number of local training samples per iteration increases as the batch size increases. In this way, the number of inter-process communications for averaging gradients per epoch decreases and it ends up having a higher scaling efficiency. The second option is to start the training with B_s active processes and increase the number of processes as the batch size increases. This design choice exploits the improved degree of parallelism. In this work, we chose the second option to employ as many workers as possible and focus on improving the scaling efficiency by overlapping the communications with the computations.

B. Overlap of Communication and Computation

In data parallel trainings, at the end of each iteration, the local gradients are averaged across all processes so that the model parameters are consistent before entering the next iteration. Intuitively, the communication for such task can be simply implemented by an MPI allreduce with the sum reduction operator. Many existing parallelizations adopt this approach [13]–[16], [21]. However, by breaking MPI allreduce into MPI all-to-all and allgather, we can achieve a better overlapping effect for averaging the gradients.

We divide the gradient averaging operations among processes, so they can be performed in parallel. In other words, each process is responsible to calculate the averages for $1/P$ of gradients. The local gradients are first redistributed among processes using an MPI all-to-all communication, so each process ends up receiving P subsets of local gradients of size $\frac{G}{P}$ each, where G is the number of gradients in the layer and P is the number of processes. Once the remote gradients are received, the P gradient subsets are element-wisely averaged into a global gradient subset of size $\frac{G}{P}$. The updated gradients are then distributed among all the processes using an MPI allgather communication. At the end, all processes obtain the same globally averaged gradients.

Breaking the allreduce into an all-to-all and an allgather has the following advantages. For a given layer k, its all-to-all can be overlapped with the computation on calculating gradients for layers $(k-1)\cdots 1$ in the back-propagation phase. Once the gradient sum for layer k is computed, an allgather is initiated, which can overlap with the computation of activations for layers $1\cdots(k-1)$ in the feed-forward phase of the next iteration. The allgather also can overlap with the gradient summation for layers $(k+1)\cdots L$, where L is the number of layers in the network. Because the number of gradients

is usually large for deep CNNs, the cost of element-wise summation is significant enough to provide more room for communication overlap. In addition, our implmentation uses the MPI-OpenMP programming model such that each MPI process parallelizes computations using OpenMP within a node. So, our approach enables the gradient summation to employ more compute cores available in each node. If the MPI allreduce approach were used, the gradients would be summed by the MPI process on a single core, losing the advantage of OpenMP multi-threading.

C. Multi-threading Implementation

We allocate one MPI process per compute node and use OpenMP on each process to utilize all the compute cores available in a node. For the matrix operations, we use Intel MKL library which efficiently utilizes KNL cores. We employ a POSIX thread per compute node to handle all the MPI communication calls. By making MPI calls in the communication-dedicated thread, we explicitly forces the overlap of the computation and the communication. The communication thread makes blocking MPI communication function calls. MPI standard defines the progress rule for asynchronous communications, but MPI implementation is free to choose whether to delay the operations till the complete functions, such as MPI_Wait and MPI_Test [32]–[34]. In some MPI implementations, we found that their asynchronous communications start only when MPI_Wait or MPI_Test is called. Due to this finding, we chose to use a communication-dedicated thread over asynchronous MPI communications.

For multi-threading management, we use POSIX thread utilities for communication between the main and communication threads. Once the gradients are computed at each layer, the main thread registers a communication request to a shared queue and sends a signal to the communication thread. Then, the communication thread receives the signal and picks a request in a first-come-first-served manner to perform the communication. Once the communication completes, the communication thread sends back a signal to the main thread to notify that the requested communication is finished. This mechanism is implemented using a pthread conditional variable and a pthread mutex. Note that contention to the mutex lock may only occur between the main and communication thread, without any OpenMP threads involved. To avoid the context switching and possible cold cache for the communication thread, we pin the communication thread on a physical core to prevent it from migrating to a different core.

V. Related Works

Recently, a few studies have shown that a large batch size can be used for classification tasks without much loss in accuracy [14], [16], [21], [22]. Layer-wise Adaptive Rate Scaling (LARS) proposed in [22] adjusts the learning rate in a layer-wise way based on the magnitude of the gradients. Our proposed training strategy can be applied to the training with LARS independently. In [22], the authors proposed a parameter update rule based on momentum SGD. Our proposed method does not affect the parameter update rule. By applying LARS at line 6 in Algorithm 1, our training strategy and LARS can be employed together without affecting each other. Therefore, when the training with LARS yields a low accuracy because of the batch size larger than the problem-dependent threshold, our proposed method can be employed and a higher accuracy can be expected.

Adaptive batch size approaches have also been proposed in [17], [23], [29]. However, these adaptive batch methods commonly control the gradient noise scale by adjusting batch size. In other words, the small batch size should be used for a sufficient number of epochs to produce a good accuracy and it significantly lowers the degree of parallelism. In [35], the authors adjust the batch size and learning rate together to increase the batch size without accuracy loss. However, their approach adjusts them based on a pre-defined schedule which should be tuned by users, making it less practical.

VI. Performance Evaluation

We evaluate the proposed parallel CNN training strategy using two image regression applications and a popular classification benchmark. We use EDSR [11] for image super-resolution with DIV2K [20] dataset and image restoration with Phantom [25] dataset. DIV2K is dataset from NTIRE2017 Super-Resolution Challenge [20], which contains 800 high-quality 2K resolution pictures. Phantom is a randomized version of the classical Shepp-Logan phantom [25], where orientation, shape and size of each of the ten ellipsoids are randomized. Phantom has 1600 training images and the size of each image is 256×256. For classification experiments, we use ResNet20 [1] and Cifar10 dataset. Cifar10 has 50,000 3-channel training images and each image size is 32×32.

Our experiments were carried out on Cori, a Cray XC40 supercomputer at National Energy Research Scientific Computing Center (NERSC). Each compute node contains an Intel Xeon Phi Processor 7250 that has 68 cores with support for 4 hardware threads each (maximum 272 threads per node). AVX-512 vector pipelines with a hardware vector length of 512 bits are available at each node. The system has the Cray Aries high-speed interconnections with 'dragonfly' topology.

We compare the performance of our proposed training algorithm with three different training methods. First, we compare our training algorithm with the best-tuned fixed batch size training. Second, as a representative fixed large batch size method, we compare our training algorithm with the linear scaling rule in [21]. Finally, we also compare with the adaptive batch size approach proposed in [17], [35]. The authors in [17] proposed to swap the learning rate decay schedule and the batch size schedule. Similarly, the authors in [35] used a pre-defined schedule for increasing the batch size.

A. Image Regression Experiments

Super-resolution is one of the classic computer vision problems, which aims to recover high-resolution images from low-resolution images [3]–[6]. Recently, many CNNs have been

TABLE I
TRAINING CONFIGURATIONS FOR DIV2K TRAINING

configurations	batch size (b)	learning rate (μ)	warmup
best batch size	16	0.0001	-
linear scaling rule	256	0.0016	gradual (5 epochs)
fixed μ, adaptive b	$64 \sim 256$	$0.0004 \sim 0.0016$	-
Proposed method	$64 \sim 256$	$0.0004 \sim 0.0016$	-

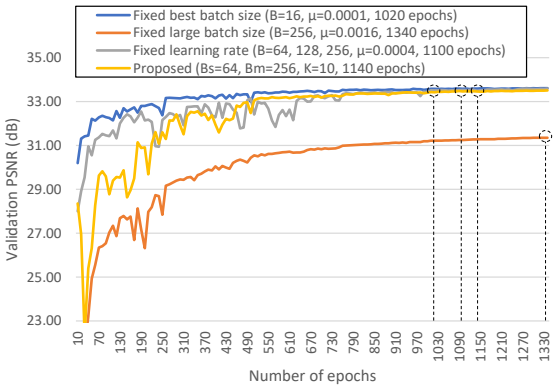

Fig. 4. Comparison of learning curves of EDSR training on DIV2K dataset among various training strategies. The proposed training method achieves an accuracy almost the same as that of the best-tuned fixed-size method.

TABLE II
TRAINING CONFIGURATIONS FOR PHANTOM TRAINING

configurations	batch size (b)	learning rate (μ)	warmup
best batch size	16	0.0001	-
linear scaling rule	128	0.0008	gradual (5 epochs)
fixed μ, adaptive b	$32 \sim 128$	$0.0002 \sim 0.0008$	-
Proposed method	$32 \sim 128$	$0.0002 \sim 0.0008$	-

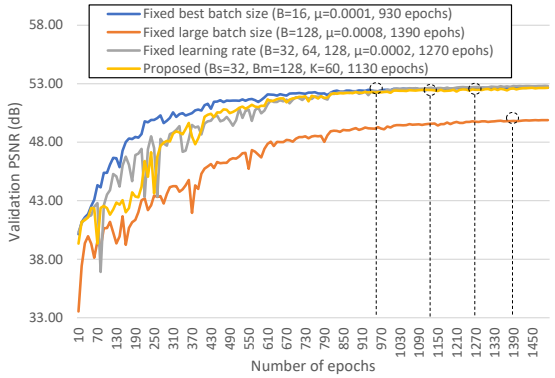

Fig. 5. Comparison of learning curves of EDSR training on Phantom dataset among various training strategies. The proposed training method achieves an accuracy comparable to the best-tuned fixed-size method, i.e. using $B = 16$.

designed for super-resolution, such as VDSR [9], DRRN [10], and EDSR [11]. Image restoration is another representative image regression problem which aims to recover original images from noisy images. Many existing works use CNNs for image denoising or compression artifact removal [3]–[6]. As we mentioned earlier, these applications typically use a small batch size between $16 \sim 64$ that gives the best accuracy. So, considering the ever-increasing available data size, it is crucial to improve the degree of parallelism by enabling the large batch size without a significant loss in accuracy.

We compare the proposed training algorithm with the same three other training methods. Table I and Table II show the training configurations for super-resolution and image restoration experiments, respectively. We use Adam for both applications. All the hyper-parameters were set to the same values as used in [11]. We use Peak Signal-to-Noise Ratio (PSNR) as the accuracy metric. PSNR measures the degree of similarity of the estimated image to the original image.

When adjusting the batch size and the learning rate in Algorithm 1 at line 12, we double them together after every K epochs because such slow increment prevents the training from diverging. Note that different increasing factors can be applied to the batch size and the learning rate such as root scaling rule depending on the problem.

For the super-resolution experiments, we randomly extract a 48×48 patch from each training image to generate mini-batches. The stop condition of the training is when the validation PSNR is not increased more than 0.1 dB for 50 consecutive epochs. The left chart in Figure 6 shows the DIV2K θ curve in the first 15 epochs. The θ peaks between $9^{th} \sim 10^{th}$ epoch. We use $K = 10$ for our proposed adaptive batch size method. We also set $B_s = 64$ since the batch size larger than 64 shows a significantly lower l2-norm. The number of DIV2K training images is 800 and 256 is the maximum power of 2 which allows more than one parameter update per epoch. So, we set $B_m = 256$.

Figure 4 compares the DIV2K learning curves among the four training methods whose configurations are given in Table I. The training with our method converges in 1100 epochs achieving a PSNR of 33.49 dB. The fixed best batch size training converges in 1020 epochs and achieves a PSNR of 33.59 dB while the large batch size training with linear scaling rule converges in 1340 epochs achieving a PSNR of 31.35 dB. The adaptive batch method with a fixed learning rate converges in 1140 epochs achieving a PSNR of 33.51 dB. Note that we calculate 1-crop validation PSNR during the training due to the significant evaluation time. So, the results can be slightly different from reported in [11].

For image restoration experiments, we use a modified EDSR which has 16 residual blocks and 32×32 input data size. Figure 5 compares Phantom dataset validation accuracy among all the training methods whose configurations are given in Table II. We found that the maximum stable batch size, B_s for Phantom is 32. The right chart in Figure 6 shows that θ peaks between $55^{th} \sim 65^{th}$ epoch ($K = 60$). Note that the training with the linear scaling rule failed to converge in a reasonable number of epochs when the batch size is larger than 128. So, to compare with other methods, we also set $B_m = 128$.

Our training strategy achieves a PSNR of 52.47 dB in 1130 epochs. The best-tuned batch size training converges in 930 epochs achieving a PSNR of 52.47 dB and the large batch size training with linear scaling rule converges in 1390 epochs achieving a PSNR of 49.84 dB. The adaptive batch size with a fixed learning rate training achieves a PSNR

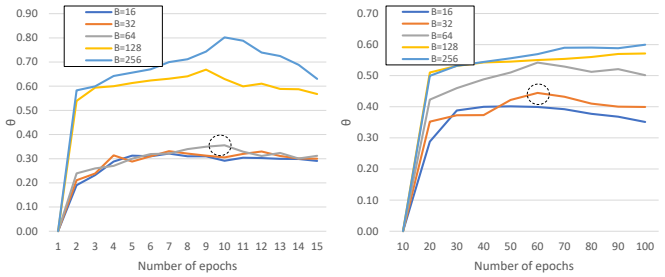

Fig. 6. The θ curves with varying batch sizes for EDSR training on DIV2K (left) and a variant of EDSR training on Phantom (right). For DIV2K, we chose $B_s = 64$ and its θ peaks at $9^{th} \sim 10^{th}$ epoch. For Phantom, we chose $B_s = 32$ and its θ peaks at $55^{th} \sim 65^{th}$ epoch.

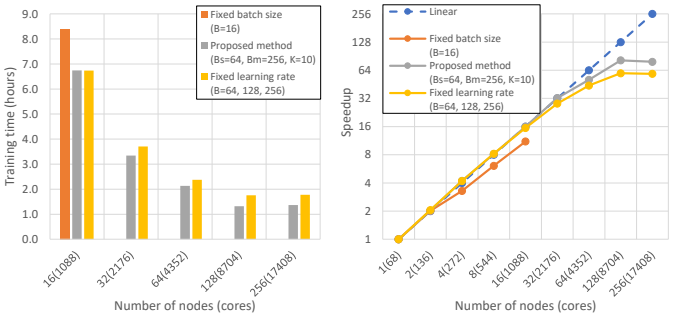

Fig. 7. Strong scaling of EDSR training on DIV2K dataset: end-to-end training time (left) and speedup (right). We used $B_s = 64$, $B_m = 256$, and $K = 10$. Our method can use more compute nodes beyond 16 and up to 256, while 'fixed best batch size' method can only run up to 16 nodes, limited by the batch size of 16. 'fixed learning rate' has a longer execution time than our method due to the long period of training with small batch sizes. All the three trainings achieve a similar accuracy (33.59 dB / 33.49 dB / 33.51 dB).

of 52.51 dB in 1270 epochs. In both super-resolution and image restoration experiments, our proposed training strategy provides comparable convergence accuracies to the best-tuned batch size training. The performance results demonstrate that the proposed adaptive batch size and learning rate method allows to use a large batch size for as many epochs as possible without a significant loss in generalization performance.

We also present the strong scaling performance to verify the effectiveness of the proposed methods. Note that we do not compare the performance against linear scaling rule methods, as they yield a significantly lower accuracy. We consider a direct comparison as unfair between two methods that produce a significantly different accuracy. Figure 7 shows the end-to-end training time (left) and the speedup (right) of EDSR training on DIV2K dataset. The best batch size training achieves a speedup of 9.54 using 16 nodes. The adaptive batch size with a fixed learning rate achieves a speedup of 71.40 using up to 256 nodes. Our proposed method achieves a maximum of speedup 81.71 and can run on up to 256 nodes. Figure 8 shows the performance of the modified version of EDSR training on Phantom dataset. The best batch size training achieves a speedup of 5.64 using 16 nodes. The adaptive batch size with a fixed learning rate achieves a speedup of 30.79 using 128 nodes. Our proposed method achieves a maximum

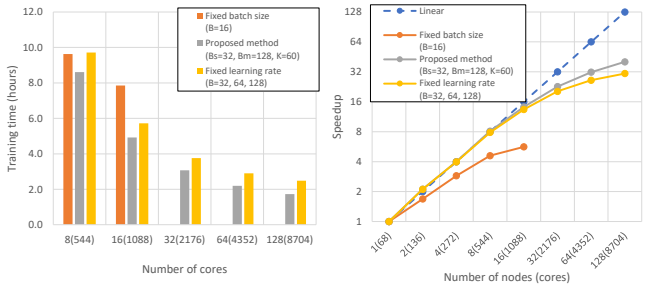

Fig. 8. Strong scaling of EDSR training on Phantom dataset: end-to-end training time (left) and speedup (right). For our proposed method, we used $B_s = 32$, $B_m = 128$, and $K = 60$. Our method can use more compute nodes beyond 16 and up to 128, while 'fixed best batch size' method can only run on up to 16 nodes, limited by the batch size of 16. All the three trainings achieve a similar accuracy (52.47 dB / 52.47 dB / 52.51 dB).

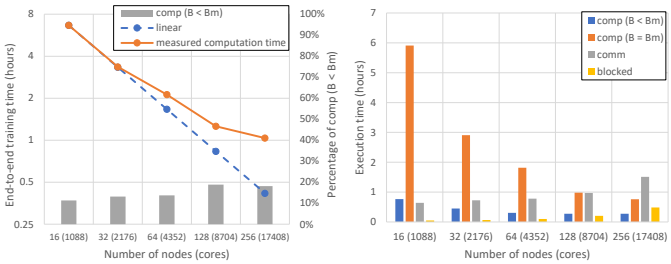

Fig. 9. The left is the computation time of EDSR training and the percentages of process underutilized time. The right is the training timing breakdown. These results correspond to the training shown in Figure 7.

of speedup 42.96 using 128 nodes. The experimental results clearly show the advantage of being able to increase the batch size. Compared to the adaptive batch size with a fixed learning rate training, our method enables to use small batch sizes for fewer epochs, which results in achieving a shorter training time as well as a higher speedup.

B. *Performance Analysis*

Communication Cost Analysis — The main reason for the speedup saturation is due to the increasing 'blocked' time, as shown in Figure 9. In a typical strong scaling result, the communication cost becomes higher and the per-process computation reduces, as the number of processes increases. When the communication is not entirely overlapped with the computation, the main thread is blocked until the communication thread finishes the transfer of data required by the main thread. For instance, when the number of processes is 128, the communication time measured at the communication thread, 'comm', grows to be similar to the main thread's computation time and 'blocked' starts to become significant. When the number of processes increases to 256, such effect becomes even more significant.

Computation Cost Analysis — Another reason for the speedup saturation is that the computation time is not linearly reduced as the number of processes increases. From the right chart of Figure 9, we observe that the computation time ('comp ($B = B_m$)') does not linearly decrease starting from 64 nodes.

TABLE III
TRAINING CONFIGURATIONS FOR CIFAR10 TRAINING

configurations	batch size (b)	learning rate (μ)	warmup
best batch size	128	0.1	-
linear scaling rule	2048	1.6	gradual (5 epochs)
fixed μ, adaptive b	$256 \sim 2048$	$0.2 \sim 1.6$	-
Proposed method	$256 \sim 2048$	$0.2 \sim 1.6$	-

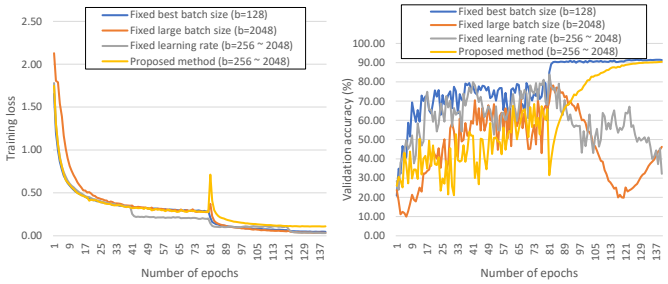

Fig. 10. Training loss (left) and validation accuracy (right) comparison for ResNet20 training on Cifar10. The fixed large batch training and the adaptive batch training with a fixed learning rate well minimize the training loss while they significantly degrade the validation accuracy. Our proposed method achieves a comparable accuracy to the best-tuned small batch training.

First, the gradient summation takes a constant amount of time regardless of the number of processes. When averaging the gradients, every process sums P gradient subsets and each is of size $\frac{G}{P}$. Thus, the computation cost for the summation is constant regardless of the number of processes. As the number of processes increases, this computation takes a larger portion of the total time. Second, when the volume of data assigned on each process is not large enough, the kernel operations, such as matrix multiplications, will not fully utilize the computation power. For example, we found that the activation computation at a convolution layer of EDSR for one sample takes $\sim$0.0026 seconds on a single KNL node while the same operation for two samples takes only $\sim$0.0038 seconds. It indicates that the hardware is underutilized when the workload is not sufficiently large, a well-known effect for KNL CPUs [36], [37]. Therefore, we suggest to assign each process at least two training samples per iteration. Third, when the batch size is smaller than the number of processes, our method replicates the training on $\frac{P}{B}$ process groups (or equivalently $(P - B)$ number of processes sitting idle). The left chart of Figure 9 also shows the percentage of computation time before the training reaches B_m from the end-to-end training time.

C. Image Classification Experiments

To verify that the proposed training method generally works for various applications, we also perform image classification experiments. We train ResNet20, one of the most popular deep CNN models, on Cifar10 dataset. Table III summarizes the training configurations. We decay the learning rate after 80 and 120 epochs with a factor of 0.1. We found the maximum stable batch size for Cifar10 is 256 ($B_s = 256$) and set $B_m = 2048$ which is the largest power of 2 smaller than 5% of the entire training samples. For the batch size of 256, θ peaks at 5^{th} epoch as shown in Figure 2, so we set $K = 5$. For the

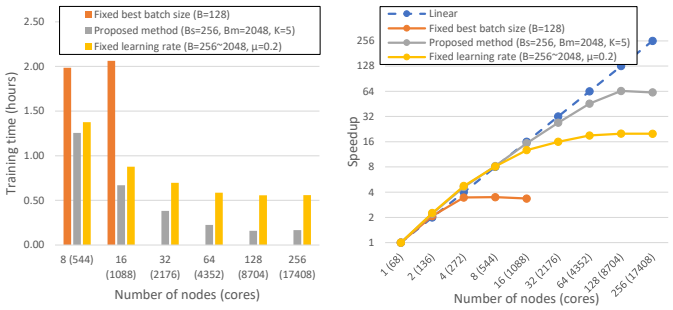

Fig. 11. The end-to-end training time (left) of ResNet20 on Cifar10 and speedup (right) comparison. We stopped scaling when the execution time increased. The proposed method out performs the others with a large margin.

adaptive batch training with a fixed learning rate, we fixed the learning rate to 0.2 and doubled the batch size from 256 after every 30 epochs so that the batch size ends up reaching 2048. Figure 10 compare the training cost curves (left) and the validation accuracy curves (right) among various training methods. Our algorithm achieves a convergence accuracy which is $< 1\%$ lower than that of the best-tuned small batch training ($91.51\% \pm 0.3$ and $90.79\% \pm 0.2$). Both the fixed large batch training and the adaptive batch training with a fixed learning rate effectively minimize the cost function, however they significantly degrade the generalization performance.

The validation accuracy comparison in Figure 10 verifies that the proposed adaptive batch size and learning rate control methods effectively increase the batch size without a significant loss in accuracy for classification problems as well. Before the first learning rate decay step, the learning curve fluctuates due to the increasing learning rate. However, the validation accuracy increases dramatically after the learning rate is adjusted at 80^{th} epoch by the proposed method.

Figure 11 presents the strong scaling performance. We stopped scaling up when the execution time increases. The training with the best-tuned batch size (128) achieves a speedup of 3.49 only due to the high ratio of communication to computation. The adaptive batch training with a fixed learning rate achieves the maximum speedup of 20.01 on 128 nodes. Our proposed method achieves a speedup of 64.64 using 256 nodes thanks to the early increase of the batch size.

VII. CONCLUSIONS

In this paper, we proposed a parallel CNN training strategy with adaptive batch size and learning rate. Our proposed method adaptively adjusts the batch size based on the estimated sharpness of the minimizer. We also presented a multi-threaded implementation of data-parallelism which enables to overlap the communication with computation. Our experimental results demonstrate that the proposed methods increase the batch size without a significant accuracy loss so that the training can scale up using more compute nodes. We believe studying practical use-cases of the proposed adaptive batch method in real-world deep learning applications 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. This work is also supported in part by DOE awards DE-SC0014330 and DE-SC0019358.

This research used resources of the National Energy Research Scientific Computing Center (NERSC), a U.S. Department of Energy Office of Science User Facility operated under Contract No. DE-AC02-05CH11231.

REFERENCES

[1] 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*, pp. 770–778, 2016.

[2] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4700–4708, 2017.

[3] C. Dong, Y. Deng, C. Change Loy, and X. Tang, "Compression artifacts reduction by a deep convolutional network," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 576–584, 2015.

[4] P. Svoboda, M. Hradis, D. Barina, and P. Zemcik, "Compression artifacts removal using convolutional neural networks," *arXiv preprint arXiv:1605.00366*, 2016.

[5] W. Dong, P. Wang, W. Yin, and G. Shi, "Denoising prior driven deep neural network for image restoration," *IEEE transactions on pattern analysis and machine intelligence*, 2018.

[6] X.-J. Mao, C. Shen, and Y.-B. Yang, "Image restoration using convolutional auto-encoders with symmetric skip connections," *arXiv preprint arXiv:1606.08921*, 2016.

[7] C. Szegedy, A. Toshev, and D. Erhan, "Deep neural networks for object detection," in *Advances in neural information processing systems*, pp. 2553–2561, 2013.

[8] S. Zhang, L. Wen, X. Bian, Z. Lei, and S. Z. Li, "Single-shot refinement neural network for object detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4203–4212, 2018.

[9] J. Kim, J. Kwon Lee, and K. Mu Lee, "Accurate image super-resolution using very deep convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1646–1654, 2016.

[10] Y. Tai, J. Yang, and X. Liu, "Image super-resolution via deep recursive residual network," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, vol. 1, p. 5, 2017.

[11] 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*, pp. 136–144, 2017.

[12] H. Robbins and S. Monro, "A stochastic approximation method," *The annals of mathematical statistics*, pp. 400–407, 1951.

[13] X. Jia, S. Song, W. He, Y. Wang, H. Rong, F. Zhou, L. Xie, Z. Guo, Y. Yang, L. Yu, *et al.*, "Highly scalable deep learning training system with mixed-precision: Training imagenet in four minutes," *arXiv preprint arXiv:1807.11205*, 2018.

[14] A. Mathuriya, D. Bard, P. Mendygral, L. Meadows, J. Arnemann, L. Shao, S. He, T. Kärnä, D. Moise, S. J. Pennycook, *et al.*, "Cosmoflow: using deep learning to learn the universe at scale," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*, p. 65, IEEE Press, 2018.

[15] T. Kurth, J. Zhang, N. Satish, E. Racah, I. Mitliagkas, M. M. A. Patwary, T. Malas, N. Sundaram, W. Bhimji, M. Smorkalov, *et al.*, "Deep learning at 15pf: supervised and semi-supervised classification for scientific data," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, p. 7, ACM, 2017.

[16] T. Kurth, S. Treichler, J. Romero, M. Mudigonda, N. Luehr, E. Phillips, A. Mahesh, M. Matheson, J. Deslippe, M. Fatica, *et al.*, "Exascale deep learning for climate analytics," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*, p. 51, IEEE Press, 2018.

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

[18] 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.

[19] L. Chen, H. Wang, J. Zhao, D. Papailiopoulos, and P. Koutris, "The effect of network width on the performance of large-batch training," in *Advances in Neural Information Processing Systems*, pp. 9322–9332, 2018.

[20] 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*, pp. 1110–1121, IEEE, 2017.

[21] 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.

[22] Y. You, I. Gitman, and B. Ginsburg, "Scaling sgd batch size to 32k for imagenet training," *arXiv preprint arXiv:1708.03888*, 2017.

[23] S. Jastrzebski, Z. Kenton, D. Arpit, N. Ballas, A. Fischer, Y. Bengio, and A. Storkey, "Three factors influencing minima in sgd," *arXiv preprint arXiv:1711.04623*, 2017.

[24] 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*, pp. 1731–1741, 2017.

[25] L. A. Shepp and B. F. Logan, "The fourier reconstruction of a head section," *IEEE Transactions on nuclear science*, vol. 21, no. 3, pp. 21–43, 1974.

[26] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[27] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[28] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[29] L. Balles, J. Romero, and P. Hennig, "Coupling adaptive batch sizes with learning rates," *arXiv preprint arXiv:1612.05086*, 2016.

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

[31] 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*, pp. 1731–1741, 2017.

[32] S. Didelot, P. Carribault, M. Pérache, and W. Jalby, "Improving mpi communication overlap with collaborative polling," *Computing*, vol. 96, no. 4, pp. 263–278, 2014.

[33] T. Hoefler, A. Lumsdaine, and W. Rehm, "Implementation and performance analysis of non-blocking collective operations for mpi," in *Proceedings of the 2007 ACM/IEEE conference on Supercomputing*, p. 52, ACM, 2007.

[34] R. Brightwell, R. Riesen, and K. D. Underwood, "Analyzing the impact of overlap, offload, and independent progress for message passing interface applications," *The International Journal of High Performance Computing Applications*, vol. 19, no. 2, pp. 103–117, 2005.

[35] A. Devarakonda, M. Naumov, and M. Garland, "Adabatch: adaptive batch sizes for training deep neural networks," *arXiv preprint arXiv:1712.02029*, 2017.

[36] I. Masliah, A. Abdelfattah, A. Haidar, S. Tomov, M. Baboulin, J. Falcou, and J. Dongarra, "High-performance matrix-matrix multiplications of very small matrices," in *European Conference on Parallel Processing*, pp. 659–671, Springer, 2016.

[37] K. Kim, T. B. Costa, M. Deveci, A. M. Bradley, S. D. Hammond, M. E. Guney, S. Knepper, S. Story, and S. Rajamanickam, "Designing vector-friendly compact blas and lapack kernels," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, p. 55, ACM, 2017.