Two Tiered Distributed Training Algorithm for Acoustic Modeling

Pranav Ladkat, Oleg Rybakov, Radhika Arava, Sree Hari Krishnan Parthasarathi, I-Fan Chen, Nikko Strom

Amazon.com

{ladkat,rybakovo,aravar,sparta,ifanchen,nikko}@amazon.com

Abstract

We present a hybrid approach for scaling distributed training of neural networks by combining Gradient Threshold Compression (GTC) algorithm - a variant of stochastic gradient descent (SGD) - which compresses gradients with thresholding and quantization techniques and Blockwise Model Update Filtering (BMUF) algorithm - a variant of model averaging (MA). In this proposed method we divide total number of workers into smaller subgroups in a hierarchical manner and limits frequent communication within each subgroup. We update local model using GTC within a subgroup and global model using BMUF across different subgroups. We evaluated this approached by training deep long short-term memory (LSTM) recurrent neural network for automatic speech recognition (ASR) problem on a 2000 hour audio dataset, and comparing to BMUF training with 128 GPUs, the proposed approach delivers 1.25x relative speed up (~100x speed up comparing to single GPU) and reduces relative WER degradation by 100%.

Index Terms:Speech recognition, Two Tiered Algorithm, distributed stochastic gradient descent, gradient threshold compression, BMUF

1. Introduction

The recent trend has shown that accuracy of the deep learning models can be improved significantly by increasing the size of training data or increasing the capacity of the model[1], [2], [3]. But as the model size and data size increase, it also causes an increase in training time. SGD is the most widely used training algorithm for training all kinds of neural networks (e.g. [4], [5]). Even if other techniques have been proposed such as alternating direction method of multipliers (ADMM) [6][7] or 2^{nd} order Hessian free optimization [8][9], SGD is still an important part of the recipe and consumes most of the time[10]. Hence, we have seen much efforts have been put into improving its efficiency and speed.

It is common practice to compute sub-gradients using mini-batches of samples rather than updating weights after each training sample and this generally gives significant boost in performance. The training time then can be further reduced by using multiple CPUs and/or GPUs and performing data parallel training. However, there are several challenges in scaling SGD to a large number of workers. First, by increasing the number of compute nodes the effective (aggregated) mini-batch size is increased linearly, which has shown to produce lower accuracy on test data [11], [12]. Several techniques can be employed such as adjusting learning rate [13], [11], using a warmup phase, etc. however these techniques increase the upper bound on workable minibatch sizes, but do not remove it. Also, increasing mini-batch size may not be possible due to limited GPU memory. Second, communicating gradients or model weights among workers is an expensive operation and its time increases rapidly as model size and number of workers are increased. This also depends on whether workers are on the same host or on a different, since communication bandwidth between different hosts is usually much lower than on the same host.

Several techniques have been proposed which addresses these fundamental limitations of increased communication time, e.g. gradient sparsification using gradient thresholding and compression using quantization techniques [14], [15], [16], [17] reduces the amount of data communicated between workers where as efficient communication algorithms have been proposed such as ring-allreduce [18], hierarchical ring-allreduce [19] to utilize bandwidth efficiently. Asynchronous variants of SGD [20] have been used which mask communication latency and improve throughput, however we will focus on synchronous variants of SGD which offers excellent reproducibility. Model parallel distributed training is another attractive option for very large models [10], however in this paper we are focused on large scale distributed data parallel training.

Model Averaging (MA) [21] is another promising technique that has been used to scale distributed training. In this approach each worker updates their local model independently using subset of dataset and then obtains global model by simply averaging independent local models. This approach can scale almost linearly to large number of workers, however achieves lower accuracy on test data and it is further affected by increasing number of workers [22][23]. There have been variants of MA proposed such as natural gradient SGD [24], BMUF [22] which improves model accuracy over simple MA. BMUF algorithm in particular has shown to achieve near linear scaling without affecting accuracy. However, from our results (shown in section 3.3.1 and 3.3.2), we noticed that as the number of workers are increased, the frequency of communication needs to be increased to achieve comparable accuracy on test dataset and this affects the scalability of the algorithm.

In this paper we introduce a hybrid algorithm which combines GTC and BMUF in a two-tiered architecture which achieves better trade-offs between accuracy and scalability.

2. Methods

2.1. Gradient Threshold Compression

This method leverages SGD with gradient thresholding and gradient quantization algorithm proposed by Strom [15]. We refer to this algorithm as GTC for brevity. In this approach, instead of sending entire gradient tensor for each trainable weight, only gradients whose absolute magnitude is greater than a predefined value, here referred as *gradient-threshold* (τ) are sent to other workers. This results in a sparse gradient update reducing total update size by couple of orders of magnitude. Each worker sends its sparse update to rest of the workers and receives their sparse updates. The received sparse gradient updates are aggregated and weights are updated with these gradients. The gradients which are not sent to other workers are stored for later iterations. In naive implementation, sparse update can be represented by two numbers, an integer element index and floating point number which is either $+\tau$ or $-\tau$. However this can be compressed further by quantizing gradient and packing quantized gradient and integer index into single 32-bit integer field. In this work, we use 1-bit quantization. Thus each worker simply send gradient deltas of $\pm\tau$ and remaining 31-bits are used for element index. It achieves further 2x compression of the data to be sent. This technique can be applied to synchronous as well as asynchronous variant of SGD, however we select synchronous variant because of its reproducible nature. This algorithm is described in more details in [15].

The synchronous variant of this algorithm achieves accuracy comparable to synchronous SGD till 64 number of GPUs as shown in 3.3.1. The gradient-threshold as well as quantization technique can be further fine tuned to achieved best trade-off between accuracy and scalability. Even though this approach improves the performance significantly over uncompressed SGD, the frequent synchronization between workers becomes a bottleneck as number of workers are increased. This bottleneck is magnified even more due to recent advancements in GPUs which takes far less computation time.

2.2. Blockwise Model Update Filtering

The BMUF algorithm [22] is a variant of MA where simple models averaging is augmented by considering model from previous step. This algorithm is split into two steps. Before the first step, the initial global model (W_g) is broadcasted to each worker. In the first step, each worker updates its local model (W) in parallel with its portion of data for a specified number of mini-batches, here referred as *block-size*. This step is referred as intra-block parallel optimization. In this implementation, each worker simply updates its local model using mini-batch SGD independently. In the second step, the global model is updated using following procedure which is referred as BMUF step.

$$\overline{W}(t) = \frac{1}{N}\sum_{i=1}^{N} W(t)^i \tag{1}$$

$$G(t) = \overline{W}(t) - W_g(t-1) \tag{2}$$

$$\Delta(t) = \eta_t \Delta(t-1) + \zeta_t G(t) \tag{3}$$

$$W_g(t) = W_g(t-1) + \Delta(t) + \eta_{t+1}\Delta(t) \tag{4}$$

where hyper-parameters η and ζ are called block momentum and block learning rate respectively. We used following formula

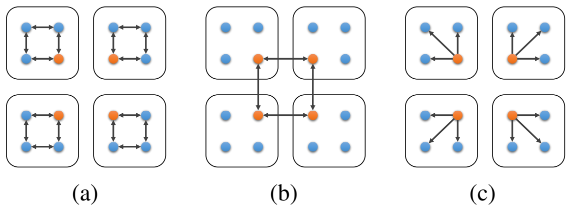
$$\frac{\zeta}{N(1-\eta)} = C \tag{5}$$

to set η and ζ hyper-parameters, where $C \geq 1$ is constant and N is number of workers. We use nesterov block momentum (NBM) scheme proposed in [22].

2.3. Proposed Two Tiered Training Algorithm

The synchronous GTC described in 2.1 suffers from scalability issues when number of workers are increased. This is especially evident in cases where ratio of compute-to-communication time is low. The primary bottleneck in this method is increasing amount of time spent in communicating gradient updates at every mini-batch. This is discussed in section 3.3.1. On the other hand, BMUF can scale almost linearly, at least in terms of throughput, with adjustment of block-size. However, as shown in the section 3.3.2, global model needs to be updated more frequently to achieve acceptable accuracy at large number of workers, which then affects the scalability of the algorithm. One of the reason of lower accuracy in BMUF algorithm is due to errors introduced from simple averaging of model weights in eq. (1) and this error increases with more number of distinct models as well as with larger block-size (less frequent communication among workers). So we hypothesize that if we reduce total number of distinct models resulting in intra-block parallel optimization step, we can achieve better accuracy without sacrificing scalability. To test this hypothesis, we propose two tiered training algorithm which combines GTC algorithm with BMUF. We refer this algorithm as $BMUF_{GTC}$ for brevity.

Figure 1: *2-Tiered Training Algorithm where N=16, P=4 1(a) workers in each sub-group performs GTC within themselves. 1(b) one worker from each group participates in BMUF step and updates global model. 1(c) The updated global model is broadcasted to rest of the workers in each subgroup.*



In this method, we divide total number of workers into M subgroups where each group contains $P = \frac{N}{M}$ workers, here referred as *group-size*. We refer to these subgroups as lower-tier. We then select one worker from each subgroup and form another subgroup, thus forming a 2-tiered hierarchy. We refer to this new subgroup as an upper-tier and it will contain M workers. All workers are initialized with same global model, however each worker processes non-overlapping subset of training dataset. Training is performed in three steps. In the first step, workers from each subgroup from lower-tier updates their model using GTC for specified number of mini-batches, here referred as block-size. Each subgroup updates their model independent to other subgroups in parallel and communication is restricted within each subgroup only. This step is also called as intra-block parallel optimization[22]. This is shown in fig. 1(a). At the end of this step workers will have updated their local models and will result into M different models, one from each sub-group. In the second step, The workers in upper tier now performs BMUF step with these M models and computes new global model using BMUF-NBM procedure mentioned in section 2.2. This step is shown in fig. 1(b). And finally in the last step, The updated global model is then broadcasted to rest of the workers in each sub-group in lower tier as shown in fig. 1(c). These three steps are repeated till all workers have completed training on their subset of dataset which is equivalent to training for 1 epoch. The training then can be continued for multiple epochs with fine-tuned hyper-parameters until a final model is obtained.

Usually workers in the same sub-group in lower tier reside on the same host to utilize faster communication channels such as shared memory (for CPUs) or peer-to-peer communication (for GPUs) needed for frequent synchronization for SGD. Workers in upper tier reside on separate hosts which only synchronizes at end of the block. This approach drastically reduces the overall communication time by restricting communication to subgroups instead of all workers. This is especially noticeable

Table 1: *Effect of number of GPUs on relative WER reduction (in %) with respect to 1-GPU SGD and relative speedup (as a factor) achieved in total frames/second and total training time until convergence. Note: block-size 50 is used for BMUF and BMUF$_{GTC}$.*

Training Method	Number of GPUs	Speedup in Frames/Sec	Speedup in Total Training Time	Relative WER Reduction (%)
GTC	16	12.5	21	0.4
	32	23.51	26.3	-1.4
	64	21.66	17.4	-2.8
	128	10.93	10.8	-15.6
BMUF	16	15.7	20.5	-0.8
	32	28.1	36.6	-3.5
	64	58.2	56.9	-8.9
	128	101.5	80.4	-9.6
BMUF$_{GTC}$	16	13.9	18.2	-0.5
	32	24.2	31	0.1
	64	49.2	42.3	-3.2
	128	97.8	97.4	-4.7

on clusters which do not provide high bandwidth interconnects such as infiniband.

3. Experiments

3.1. Hardware and Infrastructure

The experiments are carried out on Amazon Web Services (AWS) cloud infrastructure. The compute nodes used are of p3.16xlarge instance type provided by AWS Elastic Compute Cloud (EC2) service. Each compute node is equipped with 8 NVIDIA Tesla V100 GPUs and each of these GPUs offer 5,120 CUDA cores and 16 GB of device memory. The Amazon AWS EC2 P3 instances also include NVLink for ultra-fast GPU to GPU communication. Standard 25 Gbps network bandwidth is available between two compute nodes. AWS Simple Storage Service (S3) is used as data storage for external data, such as feature vectors and supervision targets.

We implemented these algorithms on in-house deep learning toolkit written in C++. The toolkit uses MPI[25] and NCCL[26] libraries for performing communication among workers. It also leverages CUDA and CuDNN libraries when running on NVIDIA devices. The 2-tiered hierarchy of workers is achieved by using communicator abstraction provided by MPI library. The datasets are pre-partitioned into multiple files and each worker fetches its portion of dataset directly from S3 in parallel. We adopt one GPU per worker strategy and spin up number of workers equivalent to total number of GPUs in the cluster.

Experiment time also includes time taken for fetching data from S3, staging mini-batch data to device along with training. Also, the single-GPU SGD baseline do not perform any gradient thresholding and quantization where as distributed version performs these extra computations and it's time is included in total training time along with cost of communication.

3.2. Experimental Setup

We benchmark proposed method on automatic speech recognition (ASR), however it can also be applied to other areas as well. Our experiments were conducted with context dependent hybrid DNN-HMM where the DNN models the posterior probabilities of phonetic states given an observation vector. We evaluate end-to-end performance in terms of Word Error Rate (WER).

For training data, we use 2000 hours of transcribed in-house Amazon Speech data. The average training utterance is about 2 seconds. First we extract 64 dimensional log filter bank energy (LFBE) feature with 10ms time shift. We then stack 3 adjacent frames to form 192 dimensional features and then down sample it by factor of 3 to form low frame rate feature setup. Frame error rates after every epoch is calculated using validation dataset where we use 1 hour of audio data and extract features using similar setup to training data. We use additional test data of 24 hours to compute WER.

The model used in current experiments consists of 5 layers of LSTM where each hidden layer is of 768 dimension, followed by an affine transform layer and a softmax layer of 3183 dimensions for predicting senone posterior probabilities. The neural network consists of approximately 24M trainable parameters which translates to approximate 93 MB model size. We feed in third frame from future to predict current output label which we found empirically to give best WER performance on training data.

To initialize model, pre-training is done on smaller dataset of 250 hours of audio data on single-GPU SGD until reasonable accuracy is achieved. For consistency, we used same pre-trained model for each experiment. We experimented with different mini-batch sizes and found that mini-batch size of 2048 frames to be optimal in terms of accuracy and GPU utilization. For learning rate scheduler, we use "min-rate newbob" scheduler in which fairly large learning rate is used initially, and model is trained until reduction in loss between two consecutive epochs is less than 5%. The learning rate is then halved for every epoch thereafter and training is stopped when learning rate reaches minimum set value. The learning rate and other hyper-parameters are carefully tuned to give the best accuracy and best results have been presented here.

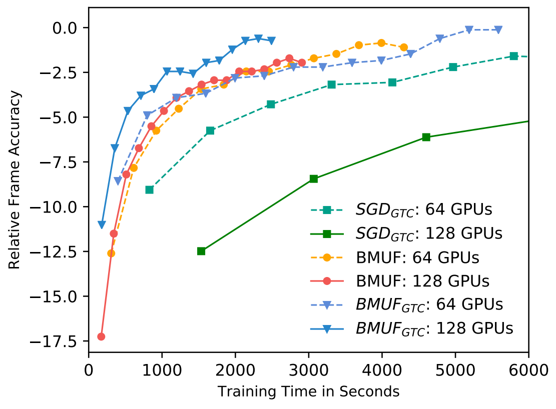


Figure 2: *Convergence of GTC, BMUF and BMUF$_{GTC}$ relative to 1-GPU SGD on 64 and 128 GPUs. Note: block-size of 50 is used for BMUF as well as BMUF$_{GTC}$, process-group-size of 8 is used for BMUF$_{GTC}$.*

For SGD_{GTC} trainer, we used gradient-threshold of 8 along with 1-bit quantization which achieved best trade-off between scalability and accuracy. For standalone BMUF as well as BMUF used in proposed method, we used Nesterov block momentum scheme for all the experiments. For hyper-parameters

Table 2: *Effect of varying block-size on relative WER reduction (in %) and training speedup (as a factor) for BMUF and BMUF$_{GTC}$ compared to 1-GPU SGD trainer on 128 GPUs.*

Training Method	Block Size	Speedup in Frames/Sec	Speedup in Training Time	Relative WER Reduction (%)
BMUF	25	91.3	91.2	-8.1
	50	101.5	80.4	-9.6
	100	106.5	64.5	-13.7
	200	110.3	57.4	-16.6
BMUF$_{GTC}$	25	77.9	101.3	-4.4
	50	97.8	97.4	-4.7
	100	99.6	80.5	-6.4
	200	105.5	73.5	-7.5

related to BMUF, we set ζ and C to 1.0 and calculate η as per (5). In proposed method, N is equated to number of subgroups (M) in (5), gradient-threshold of 2 is used along with 1-bit quantization. In all experiments group-size of 8 is used except in section 3.3.3 where effect of group-size is studied.

3.3. Results

To produce a baseline result, we trained model using single-threaded SGD on the same hardware and infrastructure. For reference, the elapsed training time of the first epoch was 290 minutes when run on single GPU. The ASR accuracy achieved by this model serves as the baseline by which we measure relative WER reduction in all other results below.

3.3.1. Effect of scaling worker nodes

To study the scaling properties of BMUF$_{GTC}$ and compare it with other methods, the number of workers were varied from 16 to 128. All other hyper-parameters are kept constant. The results of relative WER, speedup in frames/second and speedup in total training time to achieve final converged model of three methods are tabulated in table 1. Please note that different methods achieved convergence after different number of epochs and this is factored into total training time. The GTC achieves WER within 3% relative to baseline till 64 number of workers however then degrades significantly when run on 128 workers. The degradation in WER at large number of workers is mostly due to increase in effective mini-batch size. This algorithm scales satisfactorily till 32 GPUs however does not scale well beyond that. This is largely due to increased cost of communication due to increase in number of workers. The increased communication cost is mainly due to limited network bandwidth available between two separate hosts (25Gbps) as compared to high network bandwidth within same host (300Gbps) on current infrastructure. This scaling is also affected due to increased cost of decompressing gradient updates but not significantly.

BMUF on the other hand scales almost linearly in terms of frames/second processed when number of workers are increased and when adequate block-size is selected such that communication time is much lower than computation time. However, we noticed significant WER degradation beyond 32 GPUs. BMUF$_{GTC}$ algorithm tries to achieve trade-off between WER performance and scalability. The algorithm scales satisfactorily till 64 GPUs with achieving WER within of 3% relative to baseline. WER is degraded at 128 GPUs slightly however, is observed to be much more stable than BMUF as well as GTC. Figure 2 shows the frame error rate on 64 and 128 GPUs relative to single-GPU baseline after every epoch on validation dataset until model is converged.

Table 3: *Effect of varying group-size on WER reduction (in %) relative to 1-GPU SGD on 128 GPUs for block-size 25.*

Group Size	Relative WER Reduction (%)
1	-8.1
2	-7.9
4	-5.8
8	-4.4

3.3.2. Effect of varying block-size

The results in Table 1 and fig. 2 were observed using block-size 50. To study the effect of different block-sizes on BMUF and BMUF$_{GTC}$, we reran the experiments with different block-sizes on 128 GPUs and results have been tabulated in Table 2. For smaller block-size, we observe better WER performance which is expected since we're updating global model more frequently. Also it can be seen from Table 2, even when small block-size give lower speedup in terms of frames/second, the speedup in total time to converge is highest. And when block-size is increased, we see better speedup in frames/second but training continues for longer number of epochs to achieve convergence. We notice that BMUF is more sensitive to block-size parameter when compared to BMUF$_{GTC}$. The WER achieved by BMUF$_{GTC}$ at block-size 400 still outperforms WER achieved by BMUF at block-size 25.

3.3.3. Effect of varying process group size

In all the previous experiments, group-size of 8 was used in $BMUF_{GTC}$ algorithm. To see the effect on varying group-size we re-ran the experiments with different group-size on 128 GPUs and its results are tabulated in Table 3. When group-size is equal to 1, algorithm becomes equivalent to BMUF. It can be seen that WER performance of the BMUF$_{GTC}$ increases as the group-size is increased. As group-size is increased, number of distinct models are reduced which needs to be averaged as part of BMUF step (1). This reduces errors introduced due to averaging models. In our implementation we group together workers equal to number of GPUs on available on the same host and leverages fast GPU-to-GPU data transfers, however this can be further extended to multi-host scenario which can increase WER performance further.

4. Conclusion

We have presented a 2-tiered algorithm which leverages GTC and BMUF algorithms and showed that proposed method can indeed achieves better accuracy and scalability trade-off when number of workers are high. Here, the 2-tiered architecture addresses communication bottleneck issue as well as errors introduced by model averaging step in BMUF and have found to be useful where high bandwidth interconnect such as infiniband is not available. We show that, we are able to scale till 128 GPUs with 4% relative degradation after converging model. In future, we will experiment with higher group-size to scale to 128 and higher number of GPUs.

5. References

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

[2] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *CoRR*, vol. abs/1409.1556, 2014. [Online]. Available: http://arxiv.org/abs/1409.1556

[3] J. A. Trishul M, Chilimbi Yutaka Suzue and K. Kalyanaraman., “Project adam: Building an efficient and scalable deep learning training system,” *In OSDI*, pp. 571—582, 2014.

[4] G. Hinton, L. Deng, D. Yu, G. Dahl, A. rahman Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. Sainath, and B. Kingsbury, “Deep neural networks for acoustic modeling in speech recognition,” *Signal Processing Magazine*, 2012.

[5] W. Xiong, J. Droppo, X. Huang, F. Seide, M. Seltzer, A. Stolcke, D. Yu, and G. Zweig., “The microsoft 2016 conversational speech recognition system.” *arXiv:1609.03528*, 2016.

[6] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, “Distributed optimization and statistical learning via the alternating direction method of multipliers,” *Found. Trends Mach. Learn.*, vol. 3, no. 1, pp. 1–122, Jan. 2011. [Online]. Available: http://dx.doi.org/10.1561/2200000016

[7] G. Taylor, R. Burmeister, Z. Xu, B. Singh, A. Patel, and T. Goldstein, “Training neural networks without gradients: A scalable ADMM approach,” *CoRR*, vol. abs/1605.02026, 2016. [Online]. Available: http://arxiv.org/abs/1605.02026

[8] J. Martens, “Deep learning via hessian-free optimization,” in *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ser. ICML’10. USA: Omnipress, 2010, pp. 735–742. [Online]. Available: http://dl.acm.org/citation.cfm?id=3104322.3104416

[9] I. Chung, T. N. Sainath, B. Ramabhadran, M. Picheny, J. A. Gunnels, V. Austel, U. V. Chaudhari, and B. Kingsbury, “Parallel deep neural network training for big data on blue gene/q,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 6, pp. 1703–1714, 2017.

[10] J. Dean, G. S. Corrado, R. Monga, K. Chen, M. Devin, Q. V. Le, M. Z. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, and A. Y. Ng, “Large scale distributed deep networks,” in *NIPS*, 2012.

[11] P. Goyal, P. Dollár, R. B. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He, “Accurate, large minibatch SGD: training imagenet in 1 hour,” *CoRR*, vol. abs/1706.02677, 2017.

[12] T. Ben-Nun and T. Hoefler, “Demystifying parallel and distributed deep learning: An in-depth concurrency analysis,” *CoRR*, vol. abs/1802.09941, 2018.

[13] Y. You, I. Gitman, and B. Ginsburg, “Scaling SGD batch size to 32k for imagenet training,” *CoRR*, vol. abs/1708.03888, 2017.

[14] F. Seide, H. Fu, J. Droppo, G. Li, and D. Yu, “1-bit stochastic gradient descent and application to data-parallel distributed training of speech dnns,” in *Interspeech 2014*, September 2014.

[15] N. Strom, “Scalable distributed dnn training using commodity gpu cloud computing,” *In Sixteenth Annual Conference of the International Speech Communication Association*, 2015.

[16] D. Alistarh, J. Li, R. Tomioka, and M. Vojnovic, “QSGD: randomized quantization for communication-optimal stochastic gradient descent,” *CoRR*, vol. abs/1610.02132, 2016.

[17] Y. Lin, S. Han, H. Mao, Y. Wang, and W. J. Dally, “Deep gradient compression: Reducing the communication bandwidth for distributed training,” *CoRR*, vol. abs/1712.01887, 2017.

[18] P. Patarasuk and X. Yuan, “Bandwidth optimal all-reduce algorithms for clusters of workstations,” *J. Parallel Distrib. Comput.*, vol. 69, pp. 117–124, 2009.

[19] X. Jia, S. Song, W. He, Y. Wang, H. Rong, F. Zhou, L. Xie, Z. Guo, Y. Yang, L. Yu, T. Chen, G. Hu, S. Shi, and X. Chu, “Highly scalable deep learning training system with mixed-precision: Training imagenet in four minutes,” *CoRR*, vol. abs/1807.11205, 2018.

[20] B. Recht, C. Re, S. Wright, and F. Niu, “Hogwild: A lock-free approach to parallelizing stochastic gradient descent,” in *Advances in Neural Information Processing Systems 24*, J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2011, pp. 693–701.

[21] M. Zinkevich, M. Weimer, L. Li, and A. J. Smola, “Parallelized stochastic gradient descent,” in *Advances in Neural Information Processing Systems 23*, J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, Eds. Curran Associates, Inc., 2010, pp. 2595–2603. [Online]. Available: http://papers.nips.cc/paper/4006-parallelized-stochastic-gradient-descent.pdf

[22] K. Chen and Q. Huo, “Scalable training of deep learning machines by incremental block training with intra-block parallel optimization and blockwise model-update filtering,” *In ICASSP*, 2016.

[23] W. Li, B. Zhang, L. Xie, and D. Yu, “Empirical evaluation of parallel training algorithms on acoustic modeling,” *CoRR*, vol. abs/1703.05880, 2017.

[24] D. Povey, X. Zhang, and S. Khudanpur, “Parallel training of deep neural networks with natural gradient and parameter averaging,” *CoRR*, vol. abs/1410.7455, 2014.

[25] M. P. Forum, “Mpi: A message-passing interface standard,” Knoxville, TN, USA, Tech. Rep., 1994.

[26] Nvidia, “Nccl library.” [Online]. Available: https://github.com/NVIDIA/nccl,2016