INTERSPEECH 2019
September 15–19, 2019, Graz, Austria

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 limit frequent communication across subgroups. We update local model using GTC within a subgroup and global model using BMUF across different subgroups. We evaluate this approach in an Automatic Speech Recognition (ASR) task, by training deep long short-term memory (LSTM) acoustic models on 2000 hours of speech. Experiments show that, for a wide range in the number of GPUs used for distributed training, the proposed approach achieves a better trade-off between accuracy and scalability compared to GTC and BMUF.

Index Terms: Speech Recognition, Distributed Stochastic Gradient Descent, Gradient Threshold Compression, BMUF

1. Introduction

Recent trend has shown that accuracy of deep learning models can be improved significantly by increasing the size of training data or by increasing the capacity of the model [1–3]. But as the model size or data size increase, it also causes an increase in training time. SGD is the most widely used training algorithm for training neural networks (e.g. [4, 5]). Although 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, much efforts have been put into improving its efficiency and speed.

It is common practice to improve training efficiency using mini-batch SGD and using multiple compute nodes (CPUs or GPUs) to obtain further training speed up. However, several challenges still remain in scaling SGD to large number of compute nodes: increasing number of compute nodes linearly increases effective (aggregated) mini-batch size, which has shown to produce lower accuracy on test data [11, 12]. Techniques such as adjusting learning rate [11, 13] and using a warm-up phase can be employed to mitigate issues with large batch size. However these techniques increase the upper bound on workable mini-batch sizes, but do not remove it. Also, increasing mini-batch size may not be possible due to limited GPU memory. Secondly, communicating gradients or model weights among workers is an expensive operation and its time increases rapidly as a function of the model size and the number of workers.

Several techniques have been proposed which address these fundamental limitations of increased communication time. For example, gradient sparsification with thresholding and compression techniques [14–17] reduces the amount of data communicated between workers. Furthermore, efficient communication algorithms have been proposed such as ring-allreduce [18], hierarchical ring-allreduce [19] to utilize bandwidth efficiently. Asynchronous variants of SGD [10, 20] have been used which mask communication latency and improve throughput. In this paper, we will focus on synchronous variants of SGD which offers reproducibility of results. Gradient Threshold Compression (GTC) algorithm [15], in particular, has been effective in scaling up SGD; however, it does not scale well beyond few tens of GPUs [21].

Model Averaging (MA) [22] is another promising technique that has been used to scale distributed training. In this approach each worker updates their local model independently using a subset of the dataset and then computes the global model by averaging independent local models. This approach can scale almost linearly to a large number of workers. However, this tends to yield lower accuracy on test data, especially when using a large number of workers [23, 24]. There have been variants of MA proposed such as natural gradient SGD [25] and BMUF [23]; the latter improves model accuracy over simple MA. BMUF, in particular, has been shown to achieve near linear scaling; some accuracy loss, however, was noticed when trained on large number of workers [26].

In this paper, we introduce a hybrid algorithm which combines GTC and BMUF in a two-tiered architecture. Experiments show that, for a wide range in the number of GPUs used for distributed training, the proposed approach achieves a better trade-off between accuracy and scalability compared to GTC and BMUF. The rest of the paper is organized as follows: the training algorithms are described in section 2; experimental setup including infrastructure, datasets, and models are presented in section 3. We then discuss the results of the proposed method and contrast them against GTC and BMUF in section 4. Finally, we conclude the paper in section 5.

2. Distributed Training Algorithms

This section provides a review of GTC and BMUF. It then describes the proposed distributed training algorithm.

2.1. Gradient Threshold Compression (GTC)

This method leverages SGD with gradient thresholding and gradient quantization algorithm proposed in [15]. 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 a 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

http://dx.doi.org/10.21437/Interspeech.2019-1859

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 a floating point number which is either $+\tau$ or $-\tau$. However this can be further compressed by quantizing the gradient, and packing the quantized gradient and the integer index into a single 32-bit integer field. In this work, we use 1-bit quantization. Each worker simply sends gradient deltas of $\pm\tau$ and the remaining 31-bits are used for element index. It achieves further 2x compression of the data to be sent. The gradient thresholding as well as quantization technique can be further finetuned to achieve better trade-off between accuracy and scalability. While this technique can be applied to synchronous as well as asynchronous variants, we focus on the synchronous variant.

2.2. Blockwise Model Update Filtering (BMUF)

The BMUF algorithm [23] is a variant of MA where simple model averaging is augmented by considering the 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 [23].

2.3. Proposed Two Tiered Training Algorithm

The synchronous GTC described in 2.1 suffers from scalability issues when the 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. On the other hand, BMUF can scale almost linearly, at least in terms of throughput, with adjustment of *block-size*. However, the global model needs to be updated more frequently to achieve acceptable accuracy with large number of workers, which then affects the scalability of the algorithm. One of the reasons for reduced accuracy is the mismatch in optimization introduced from simple averaging of model weights in eq. (1). This mismatch then increases with more number of distinct models as well as with larger *block-size* (less frequent communication among workers). To address this issue, we propose a hybrid two-tiered training algorithm which combines GTC with BMUF. We refer to this algorithm as Hybrid Two-Tier Method (HTM) for brevity.

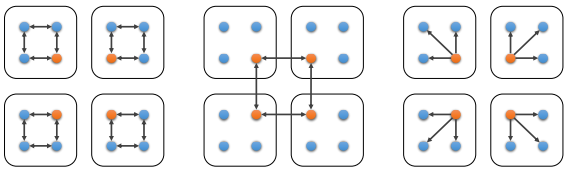

Figure 1: *Hybrid Two-Tier Method where N=16, P=4. (a) workers within each sub-group updates model using GTC. (b) one worker from each group participates in BMUF step and updates global model. (c) The updated global model is broadcasted to rest of the workers in each subgroup.*

In HTM algorithm, we update local model using GTC in intra-block parallel optimization step and update global model using BMUF-NBM procedure. We divide the 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 the same global model; however, each worker processes non-overlapping subset of the training dataset. Training is performed in three steps. In the first step, workers from each subgroup from lower-tier update their model using GTC for specified number of mini-batches, here referred as *block-size*. Each subgroup updates their model independent to other subgroups, and the communication is restricted within each subgroup. This step is also called as intra-block parallel optimization[23]. This is shown in fig. 1(a). At the end of this step workers will have updated their local models and will result in M different models, one from each sub-group. In the second step, the workers in upper tier perform BMUF step with these M models and compute the new global model using BMUF-NBM procedure (described 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 steps are repeated until a final model is obtained. This proposed method generalizes BMUF which is a special case when *group-size* is set to 1.

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 of gradients for GTC which is done every mini-batch. Workers in upper tier usually reside on separate hosts which only synchronize at end of the block (after *block-size* number of mini-batches). This approach drastically reduces the overall communication time by restricting communication to subgroups instead of all workers. This is especially noticeable on clusters which do not provide high bandwidth interconnects such as Infiniband.

3. Experimental Setup

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

Table 1: *Effect of scaling workers (GPUs) with respect to 1-GPU SGD in terms of (a) relative WERR (in %) (b) relative speedup (as a factor) achieved in total frames/second; and (c) total training time until convergence. Note: block-size 50 is used for BMUF and HTM.*

Training Method	Number of GPUs	Speedup in Frames/Sec	Speedup in Convergence Time	Relative WERR (%)
GTC	16	12.5	21	0.4
	32	23.5	26.3	-1.4
	64	21.7	17.4	-2.8
	128	10.9	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
HTM	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

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[27] and NCCL[28] 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.

3.2. Evaluation, Datasets and Models

We benchmark proposed method on automatic speech recognition (ASR), however it can also be applied to other areas as well. We report accuracy as relative word error rate reduction (WERR) (cf. [21, 26, 29–32]). Negative WERR values represent degradation in word error rate.

For training data, we use 2000 hours of transcribed in-house Amazon Speech data. The average training utterance length is about 2 seconds. 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 subsample it by factor of 3 to form low frame rate feature setup. Frame error rates after every epoch is calculated on a validation dataset consisting of 1 hour of speech data. For evaluating the models, we use an additional test data of 24 hours. For decoding, we use a 4-gram statistical language model with an acoustic model scale factor that was tuned on the test set.

Our experiments were conducted with hybrid LSTM-HMM system, where the LSTM models senone posterior probabilities given a feature vector. The model used in current experiments consists of 5 layers of LSTM where each hidden layer is of 768 dimensions, 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 approximately 93 MB model size. For training, we use cross-entropy loss function. We do a look ahead of three frames to predict current output label which we found empirically to give best WERR on training data. In this paper, since the focus is on distributed training algorithms, we do not perform sequence discriminative training.

Experiment time includes time taken for fetching data from S3, staging mini-batch data to device along with training. Single-GPU SGD baseline does 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.

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.

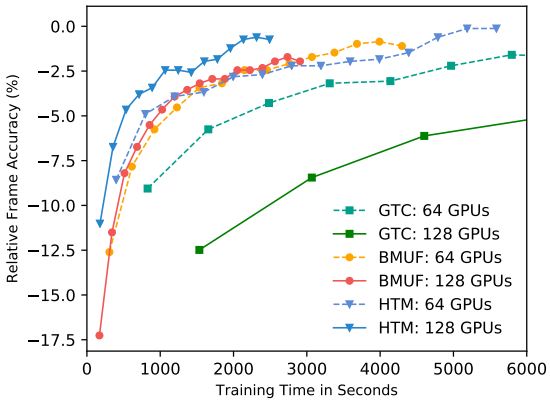

Figure 2: *Convergence of GTC, BMUF and HTM relative to 1-GPU SGD on 64 and 128 GPUs. Note: block-size of 50 is used for BMUF as well as HTM, process-group-size of 8 is used for HTM.*

For 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 related to BMUF, we set ζ and C to 1.0 and calculate η as per eq. (5). In proposed method, N is equated to number of subgroups (M) in eq. (5), gradient-threshold of 2 is used along with 1-bit quantization. In all experiments group-size of 8 is used except in section 4.3 where effect of group-size is studied.

4. Results

In this paper, since we are comparing distributed training algorithms (for training LSTM AM), our baseline model is trained

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

Training Method	Block Size	Speedup in Frames/Sec	Speedup in Convergence Time	Relative WERR (%)
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
HTM	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

using the single threaded SGD algorithm executed on the same hardware and infrastructure. For the baseline model, the elapsed training time for the first epoch was 290 minutes. The WER achieved by this model serves as the reference against which we measure relative WER reduction for all other results.

4.1. Effect of scaling workers

To investigate the scaling properties of HTM and compare it with other methods, the number of workers were varied from 16 to 128. All other hyper-parameters are kept constant. We compare the results of the three methods in Table 1 against a single GPU SGD model in terms of: (a) relative WER reduction; (b) speedup in frames/second; and (c) speedup in total training time to achieve final converged model. Please note that different methods achieved convergence after different number of epochs and this is factored in the total training time.

GTC stays within 3% relative WERR to baseline till 64 workers; it then degrades significantly when run using 128 workers. The degradation in WER at large number of workers is mostly due to the 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 the increased cost of communication due to the 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 bottleneck is magnified even more due to recent advancements in GPUs which take far less computation time. The scaling of this algorithm is also affected due to increased cost of decompressing gradient updates; but that is not a significant factor.

BMUF on the other hand scales almost linearly in terms of frames/second processed when the 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. HTM algorithm achieves a trade-off between WERR and scalability. The algorithm scales satisfactorily till 64 GPUs, achieving a WER within 3% to the baseline. It degrades at 128 GPUs slightly, however; it 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 the single-GPU baseline after every epoch on the validation dataset until after each model 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 WERR (%)
1	-8.1
2	-7.9
4	-5.8
8	-4.4

4.2. 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 HTM, we ran experiments with different block-sizes on 128 GPUs; the results are tabulated in Table 2. For smaller *block-size*, we observe improved WERR which is expected since we are updating global model more frequently. Also it can be seen from Table 2 that even when small *block-size* yields lower speedup in terms of frames/second, the speedup in total time to converge can be significant. When the *block-size* increases, we see better speedup in frames/second but training continues for more number of epochs to achieve convergence. We notice that BMUF is more sensitive to *block-size* parameter when compared to HTM. The WER achieved by HTM at a *block-size* of 200 still outperforms WER achieved by BMUF at a block-size of 25.

4.3. Effect of varying group-size

In all the previous experiments, a *group-size* of 8 was used in the HTM algorithm. To see the effect on varying the *group-size* we ran experiments with different *group-size* on 128 GPUs. The results are reported in Table 3. When *group-size* is equal to 1, the algorithm is equivalent to BMUF. It can be seen that WERR of HTM increases as *group-size* is increased. As *group-size* is increased, the number of distinct models that needs to be averaged as part of BMUF equation (1) are reduced. This reduces mismatches introduced due to averaging models. In our implementation we group together workers equal to number of GPUs available on the same host and leverage fast GPU-to-GPU data transfers. This can be extended to multi-host scenario, which can further improve WERR.

5. Conclusion

We have presented a 2-tiered algorithm which leverages GTC and BMUF algorithms and showed that proposed method can indeed achieve a better accuracy and scalability trade-off over a wide range of GPU workers. Here, the 2-tiered architecture addresses the communication bottleneck issue as well as the optimization mismatch issue introduced by the model averaging step in BMUF. It is found to be useful where high bandwidth interconnect such as Infiniband is not available. Our experiments show that, at model convergence, the proposed algorithm can scale to 128 GPUs with only 4% relative degradation against a single GPU SGD trained LSTM acoustic model.

6. Acknowledgments

We would like to thank Yasser Ibrahim and his team for supporting experimentation efforts, Gautham Kollu, Yusuf Goren, Vladimir Oster, Tianjun Ye for optimizing the toolkit and Brian King for providing initial setup for experiments.

7. 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] S. H. K. Parthasarathi, N. Sivakrishnan, P. Ladkat, and N. Strom, "Realizing petabyte scale acoustic modeling," in *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 2, 2019, pp. 422–432.

[22] 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

[23] 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.

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

[25] 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.

[26] S. H. K. Parthasarathi and S. Nikko, "Lessons from building acoustic models with a million hours of speech," in *Proc. of ICASSP*, 2019.

[27] M. P. Forum, "MPI: A message-passing interface standard," Knoxville, TN, USA, Tech. Rep., 1994.

[28] Nvidia, "NCCL library." [Online]. Available: https://github.com/NVIDIA/nccl,2016

[29] S. Garimella, A. Mandal, N. Strom, B. Hoffmeister, S. Matsoukas, and S. H. K. Parthasarathi, "Robust i-vector based adaptation of DNN acoustic model for speech recognition," in *Proc. of Interspeech*, 2015.

[30] B. King, I.-F. Chen, Y. Vaizman, Y. Liu, R. Maas, S. H. K. Parthasarathi, and B. Hoffmeister, "Robust speech recognition via anchor word representations," in *Proc. of Interspeech*, 2017.

[31] S. H. K. Parthasarathi, B. Hoffmeister, S. Matsoukas, A. Mandal, N. Strom, and S. Garimella, "fMLLR based feature-space speaker adaptation of dnn acoustic models," in *Proc. of Interspeech*, 2015.

[32] L. Mosner, M. Wu, S. H. K. Parthasarathi, R. Maas, A. Raju, K. Kumatani, S. Sundaram, and B. Hoffmeister, "Improve noise robustness of automatic speech recognition via teacher-student learning," in *Proc. of ICASSP*, 2019.