TWO-STAGE ASGD FRAMEWORK FOR PARALLEL TRAINING OF DNN ACOUSTIC MODELS USING ETHERNET

Zhichao Wang, Xingyu Na, Xin Li, Jielin Pan, Yonghong Yan

The Key Laboratory of Speech Acoustics and Content Understanding,
Institute of Acoustics, Chinese Academy of Sciences

ABSTRACT

Deep neural networks have shown significant improvements on acoustic modelling, pushing state-of-the-art performance in large vocabulary continuous speech recognition (LVCSR) tasks. However, training DNNs is very time-consuming on scaled data. In this paper, a data-parallel method, namely two-stage ASGD, is proposed. Two-stage ASGD is based on asynchronous stochastic gradient descent (ASGD) paradigm and is tuned for GPU-equipped computing cluster connected by 10Gbit/s Ethernet other than Infiniband. Several techniques, such as hierarchical learning rate control, double-buffering and order-locking are applied to optimise the communication-to-transmission ratio. The proposed framework is evaluated by training a DNN with 29.5M parameters using a 500-hours Chinese continuous telephone speech data set. By using 4 computer nodes and 8 GPU devices (2 devices used in each node), a 5.9 times acceleration is obtained over a single GPU with acceptable loss of accuracy (0.5% in average). A comparative experiment is done to compare the proposed two-stage ASGD with the parallel DNN training systems reported in prior work.

Index Terms— Speech recognition, deep neural network, asynchronous stochastic gradient descent, parallel training

1. INTRODUCTION

Recently, automatic speech recognition (ASR) systems using deep neural network (DNN) based acoustic models have shown great power in both research and industry [1, 2]. The discriminability of DNN is improved when the network grows deeper, and the generality of DNN is promised when the scale of training data increases [3]. However, training deep networks with large scale of data is time-exhausting. The reason is that the training algorithm, i.e. the stochastic gradient descent (SGD), is computationally expensive and is difficult to be parallelized due to its sequential nature. The employment of hardware acceleration improves the training speed, such as the use of GPUs, but the training of a deep network using a large scale of data is still unacceptably slow, even on a high performance computing node.

There has been much effort at parallelizing the training of DNN over a cluster of high performance computing nodes. The proposed methods mainly fall into three categories.

1. *Data parallelism*

   In data parallelism methods, the training data is spread over the cluster. Each node works on a portion of the whole data, and develops its own version of the network independent of other nodes. A master node controls how the nodes collaborate, either asynchronously (ASGD) [4] or synchronously (SSGD) [5].

2. *Model parallelism*

   In model parallelism methods, the DNN parameters are distributed across several nodes. The nodes communicate with each other by transmitting the activations when necessary [6]. A variation of such methods is the using of multiple small DNNs to compute the posterior probabilities of a subset of the targets, which solves the communication problem by moderate performance degradation [7].

3. *Hybrid of data and model parallelism*

   A well recognised hybrid system is the DistBelief [6]. Besides, parallelism can be carefully designed by exploiting the advantages of individual method for specific network topologies, such as the convolutional networks [8].

In this paper, we proposed a framework named two-stage ASGD which is used to parallelize the training using a cluster of workstations equipped with GPUs. The proposed framework falls into the first category. In conventional data parallel methods, the working nodes communicate with the master by transmitting their own version of the network, i.e. the parameters or the gradients. Although the communication cost can be partly relieved by using techniques as model shrinking [9] or

Thanks to the National Natural Science Foundation of China (Nos. 11461141004, 91120001, 61271426), the Strategic Priority Research Program of the Chinese Academy of Sciences (Grant Nos. XDA06030100, XDA06030500), the National 863 Program (No. 2012AA012503) and the CAS Priority Deployment Project (No. KGZD-EW-103-2) agency for funding.

1-Bit compression [10], these methods require the support of extra high speed networking techniques, such as Infiniband. We propose to customize the ASGD parallelism by defining master, working nodes and GPU devices. The master and nodes are connected through Ethernet. Due to the relatively low bandwidth of Ethernet, communications between master and nodes is handled by the two-stage framework such that computation-to-transmission ratio is optimized.

The rest of the article is organized as follows. Conventional ASGD method and its implementation are briefly introduced in section 2. In section 3 the proposed framework and relating optimizing techniques are presented. Section 4 and 5 give the experimental results and analysis.

2. ASYNCHRONOUS STOCHASTIC GRADIENT DESCENT AND GPU IMPLEMENTATION

Mini-batch SGD is commonly used for the fine tuning of DNNs. Parameters are updated using the gradient collected from a batch of training samples.

$$\hat{w} = w - \eta \Delta w \tag{1}$$

where w and $\hat{w}$ represent the parameters before and after updating. η is the learning rate which is used to scale the gradient Δw such that the parameters converge to the optima. The whole training set is divided into multiple mini batches, which are sequentially presented to the network for calculating the gradients.

In ASGD-based data parallel training paradigm, multiple mini batches are presented to the network simultaneously such that multiple gradients are calculated in parallel to develop multiple replicas of the network. The replicas asynchronously fetch parameters w and push gradients Δw to a parameter server. Using this method, a huge cluster of CPUs could be used to train DNNs for speech recognition faster than a GPU [6].

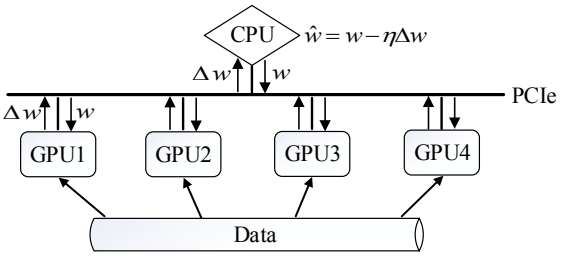

Fig. 1. Implementation of ASGD on a computing node equipped with 4 GPU devices.

This method came with acceleration guarantees [11] so it is straightforward to be implemented using multiple GPUs to scale the DNN-based acoustic modelling. In [4], a computing node is equipped with 4 GPUs and ASGD is implemented such that each GPU holds a model replica. The framework is shown in Figure 1. The CPU of the node works as the parameter server, which is responsible for memorising the freshest model and accumulating the gradients sent by GPU devices through PCIe.

3. TWO-STAGE ASGD USING GPU CLUSTER

Further scaling of DNN-based acoustic model training requires involving more GPU devices for parallelization. Considering the capacity and safety of building computing nodes, such as the limitation brought by the power supply and cooling systems, the optimal number of GPU devices installed in one node is no more than 6, which necessitates customising the training paradigm on a cluster of computing nodes, instead of one. However, due to bandwidth limitation and transmission latency of Ethernet, naive implementation of ASGD on such a cluster is impossible. The computation speed of GPU is much faster than a CPU, so that the communication between master and GPU devices can be easily jammed by massive transmission of parameters and gradients. Therefore, current parallelization on multiple GPU nodes replace Ethernet with Infiniband for high transmission speed and low latency [12, 10, 13].

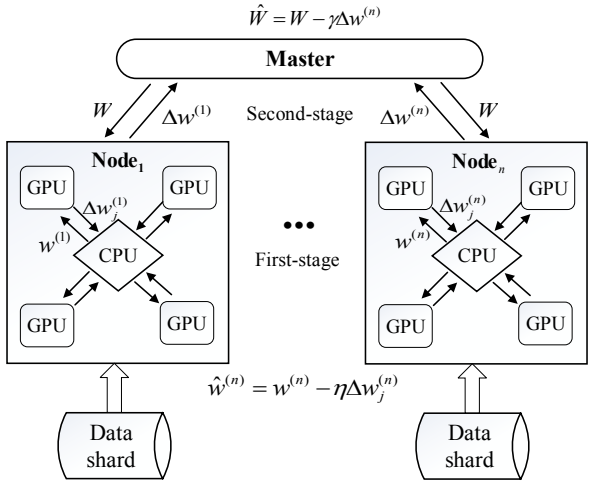

Fig. 2. Two-stage ASGD framework implemented on a cluster of GPU-equipped computing nodes.

The proposed framework, called two-stage ASGD, aims at parallelizing the training of DNNs on multiple GPU nodes using Ethernet. The framework is shown in Figure 2. Inspired by MapReduce, two-stage ASGD store data shards locally to avoid massive data transmission. Each node works on its own DNN replica, using a lock-free ASGD system as described in section 2, and communicates with the master independently using Ethernet. When N mini batches have been presented,

the aggregated gradients are pushed from the node to the master.

$$\hat{w}^{(n)} = w^{(n)} - \eta \Delta w_j^{(n)} \tag{2}$$

$$\Delta w^{(n)} = \sum_{j=1}^{N} \Delta w_j^{(n)} \tag{3}$$

Equation (2) shows the first-stage ASGD in node n. $w^{(n)}$ represents the model replica stored in that node. The GPU devices asynchronously updates the replica using gradients calculated from their own mini batches fetched from the data shard. For instance, $\Delta w_j^{(n)}$ represents the j-th update. When N updates are done, the aggregated gradient $\Delta w^{(n)}$ is ready for a push.

In the second-stage ASGD, when the master receives a pushed gradient, the model parameters are locked, updated, and sent back to the node.

$$\hat{W} = W - \gamma \Delta w^{(n)} \tag{4}$$

where W is the model parameters hosted by the master, and $\hat{W}$ is the updated version. γ is the learning rate used in the master. While the first-stage ASGD works on PCIe, the second-stage ASGD works on Ethernet. Although the two-stage framework significantly reduce the communication frequency, the communication cost is still unacceptable. Three key techniques for stablizing the system are described below.

3.1. Order-locking ASGD

The pushing period N is a scaling factor of mini batch size for the master and nodes. With bigger N, the communication frequency can be significantly reduced. However, as it is essentially an increasing of the mini batch size, the model replica may have a greater risk of divergence. Therefore, the learning rate in second-stage ASGD is shrank by the pushing period.

$$\gamma = \frac{1}{N}\eta \tag{5}$$

Due to performance difference of the nodes and network latency, gradients from different nodes might arrive in different order in each iteration [1]. To avoid the stale gradient problem, we introduce an order-locking constraint to the second-stage ASGD. At the beginning of each iteration, the master memorises the order that the gradients arrive, and keeps the gradient fetching order locked during the rest of the iterations.

3.2. Double buffering

During the transmission for the second-stage ASGD, GPUs have to stop computing and wait for the new model replica, resulting in resource waste. For example, if our system consists of one master and four computing nodes. Each node communicates with the master by transmitting DNN model parameters or gradients, whose size is usually more than 100MB in speech recognition tasks. Assuming that in the ideal condition, we take full use of the 10Gbit/s bandwidth of the Ethernet, it will take more than 640 milliseconds to transmit the parameters.

In order to avoid wasting GPUs computation resource during transmission, the double buffering mode is applied. GPUs will keep computing gradients with the old model replica while exchanging parameters between nodes and master. Although the delay between the model and the gradient becomes more severe, the delayed update has been proven to work well [14].

3.3. Computation-to-transmission ratio optimization

The computation-to-transmission ratio of the system is influenced by several factors. Firstly, bandwidth and transmission delay limit the maximum number of nodes, which limits the scale of system. To make the system efficient, the upper limit of the number of nodes is the one that saturates the bandwidth. For instance, when Message Passing Interface (MPI) is used for inter-node data transmission, communication between two nodes through a single process consumes about 40% of the bandwidth (based on our preliminary test). Thus the system can bear at most 3 nodes to communicate with the master simultaneously. Besides, the master can be used as an additional local node which does not compete for the network bandwidth. Therefore, a cluster with more than 4 nodes will not bring more benefits as network competition problem becomes critical and each node occupies less bandwidth.

Secondly, the pushing period N determines the efficiency of the system. The optimal N is to keep the gradients computation time equal to the parameters transmission time, in which way the system works most efficiently with no waste of resources. The transmission time, defined as the total time for gradient pushing and parameter fetching, is calculated using the number of nodes in the cluster and the acoustic model size.

$$T_{trans} = \frac{2N_e M}{B} \tag{6}$$

where N_e is the number of executive nodes and M denotes the model size. B represents the actual bandwidth utilised during transmission.

The computation time is decided by:

$$T_{calc} = \frac{P}{T_f} \cdot \frac{N_c N}{G_n} \tag{7}$$

where P is the number of model parameters, T_f represents the computation speed of GPU (FLOPS), and G_n is the number of GPU used in each node. N_c is the mini batch size on a computing node. In the proposed framework, each GPU device works on a mini batch of 1024 samples at one time. Once

[1] One iteration means all the training data are presented to the system

N_c samples are presented, the model replica on that node is broadcast to all the devices to avoid stale gradients. Increasing N_c and N means the system can be parallelized by using more GPUs in one node, the computation on one node will be faster. However, the DNN model on master may get unstable. In our preliminary experiments, we seek for a balance between training speed and convergence performance.

4. EXPERIMENTS

A 500-hour Mandarin conversational telephone speech dataset is used to evaluate the scalability and recognition performance of the proposed two-stage ASGD framework. 90% of the total 500-hour data is used as training set and the held-out 10% is used as the validation set for modifying the learning rate and monitoring the training. An 1-hour test set collected in real application (CTS) and an 1.5-hour test set which is a part of HKUST Mandarin Telephone Speech dataset (LDC2005S15/LDC2005T32) developing set (LDC) are used to evaluate recognition performance in character error rate (CER). The number of frames processed per second (fps) is used to evaluate the training speed of DNN. Training speech are represented by a 60-dimension feature which is formed by appending pitch and voicing strength [15] to 13-dimensional PLP [16] and their first three orders of dynamics. Concatenated features of 11 frames with a context window of 5 are used as the input of DNN. Each DNN has 5 hidden layers of 2048 nodes and an output layer with 6245 nodes representing the tri-phone states. The total number of parameters is about 29.5 million.

4.1. System optimisation for scalability

The hyper-parameters in Equation (6) and (7) need customisation for improving the scalability of the paradigm. The bandwidth B is optimised by using the MPI protocal for transmission in the 10Gbit/s Ethernet. The model scale is determined by $P \approx 29.5$million and $M \approx 112$Megabytes. T_f is fixed because all the GPU devices in our cluster are of the same model. There are 4 computing nodes in the cluster, meaning that $N_e = 4$. In a preliminary experiment on single node ASGD, the node batch size N_c is set to 4096 to produce reliable gradients. Therefore, the hyper-parameters to be tuned are the pushing period N and the number of GPU devices per node G_n.

As mentioned in [5], smaller learning rate and good initial model endorse a larger limit of the mini batch size. Table 1 shows the seek for optimal batch size N_cN by increasing N. Learning rate is modified using the strategy as Equation (5). Besides, the model ran on a warming start and the learning rate for each run decrease proportionally to N_cN. The recognition performance degrades sharply when the batch size is more than 24k.

Table 1. The influence of batch size to the final recognition results.

N_cN	4096	8192	16384	24576	32768
N	1	2	4	6	8
CER	35.3	35.2	35.3	35.4	36.3

Table 2. Training speed in 1000 frames per second (kfps) when increasing GPU devices number together with pushing step N, with and without double-buffering (DB). 4x2 means 4 nodes and 2 GPU devices for each are used in the training.

GPUs	4x1	4x2	4x3	4x4
N	2	4	6	8
w/ DB	28.2	56.3	84.6	112.5
w/o DB	15.1	30.0	44.5	59.6

The next step is to choose the optimal G_n so that $T_{calc} \approx T_{trans}$, where the system works at the maximum speedup. It is implied from Table 2 that increasing the pushing period N when employing more GPUs can speed up the training for both systems with or without double-buffering. And it is obvious that double-buffering is essential to the two-stage ASGD system.

To measure the scalability, the training is ran several times by varying the number of GPUs. Figure 3 reports the average speed in term of 1000 frames per second (kfps) using each configuration together with the final frame accuracy on the cross validation dataset. By using the system optimisation procedures, it is shown that the training speed increases almost linearly with more GPUs added. However, when using more than 8 GPUs, the frame accuracy get a sharp decline since the batch size of master is too large.

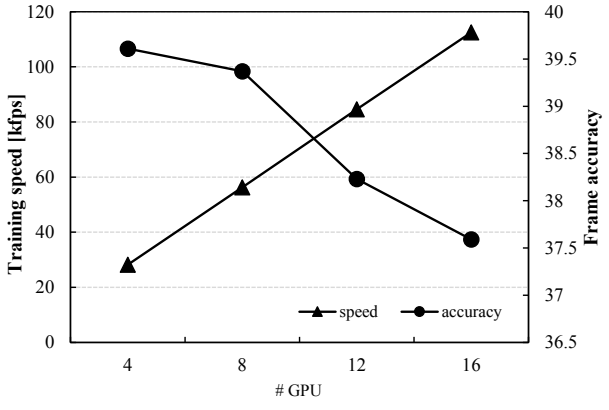

Fig. 3. Training speed and frame accuracies according to various numbers of GPUs added into the system.

Table 3. Performance comparison in terms of CER (%) and processing 1000 frames per second.

System	LDC	CTS	kfps	Speedup
Baseline	44.1	34.8	9.4	1x
Two-stage (4x2)	44.6	35.4	55.7	5.9x

4.2. Recognition performance

In the performance comparison experiment, a baseline DNN is trained using a single GPU card with mini batch of 1024 frames and random initialisation. We use a decaying learning rate strategy that starts from 0.008 and halves after each iteration once the frame accuracy on the validation set increase less than 0.5% for three times. The first iteration of the baseline is used as a warming start initial model. In the fine tuning process, all 4 nodes with two GPUs on each are used. The configuration of learning rate of each client is the same as the baseline. Therefore, G_n is 2 and N is set to 6. Table 3 shows the performance comparison of the proposed two-stage ASGD system and the baseline in CER and training speed (the initialization time is included). We can see the proposed two-stage ASGD system can give a 5.9 times acceleration with slight performance degradation, 0.5% absolutely.

4.3. Comparison with prior work

This paper focus on finding a method to take use of the Ethernet to apply distributed training of DNN with multiple GPU devices while prior approaches usually require the support of InfiniBand. The framework is inspired by the idea of ASGD and customised for fine tuning of DNN models using GPU cluster. Table 4 compares the proposed method with some approaches reported in prior work, such as single-ASGD [4], multi-DNN [7] and 1-Bit SGD [10]. Since the test sets are different, we take speedup and relative C/WER loss according to corresponding baseline systems as consideration. Compared with single-ASGD, the proposed framework has a better scalability since the single-ASGD could not use more than one computing node. Multi-DNN split the original DNN into several smaller pieces and each piece is trained independently. Although this method leads to a rather high efficiency in speedup, a relative high WER loss is observed. From the last two rows, it is shown that the proposed framework performs slightly better than 1-Bit SGD system with InfiniBand when using the same number of GPUs.

5. CONCLUSION

In this paper, an asynchronous parallelization framework for training DNNs, namely two-stage ASGD, is proposed. The proposed framework allows training deep neural networks with multiple GPU-equipped computing nodes connected across Ethernet. Several techniques to optimise the training system are introduced, such as hierarchical learning rate control, order-locking and double-buffering. Using two-stage ASGD, the bandwidth bottleneck problem in Ethernet-based cluster is addressed. Experimental results show that the proposed framework achieves significant speedup by using multiple GPUs on multiple nodes. The scaling on GPU cluster is almost linear. In the future, we will investigate introducing more advanced learning rate modification technology, such as AdaGrad [17], to further scale the training system with Ethernet. For instance, [10] used 40 GPU devices with support from Infiniband.

Table 4. Comparison with prior work in terms of speedup factor and C/WER loss. IFB means whether Infiniband is used.

System	IFB	#GPU	Speedup	C/WER loss
ASGD[4]	No	4	3.2x	1.8%
MultiDNN[7]	No	4	4.9x	5.4%
1-Bit[10]	Yes	8	5.6x	1.8%
Two-stage	No	8	5.9x	1.4%

6. FUTURE WORK

We have verified the validity of the proposed the framework by training a classical DNN using cross entropy criterion. In the future work, we will explore the universality of the method to DNN sequence training with utterance-level loss functions. As mentioned in 3.1, the second-stage ASGD is strictly order-locked and the length of the utterances processed by computing nodes determines the length of computation time. To balance the computation time of each node, the training data may be firstly ranked by the length of the sentence so that each node deal with the same amount of frames at one time.

7. REFERENCES

[1] Geoffrey E. Hinton, Li Deng, Dong Yu, George E. Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N. Sainath, and Brian Kingsbury, "Deep neural networks for acoustic modeling in speech recognition," *IEEE Signal Processing Magazine*, pp. 82–97, November 2012.

[2] George E. Dahl, Dong Yu, Li Deng, and Alex Acero, "Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 20, pp. 30–42, January 2012.

[3] Abdel-rahman Mohamed, George E. Dahl, and Geoffrey Hinton, "Acoustic modeling using deep belief networks," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 20, pp. 14–22, January 2012.

[4] Shanshan Zhang, Ce Zhang, Zhao You, Rong Zhen, and Bo Xu, "Asynchronous stochastic gradient descent for DNN training," in *ICASSP*, 2013, pp. 6660–6663.

[5] Frank Seide, Hao Fu, Jasha Droppo, Gang Li, and Dong Yu, "On the parallelizability of stochastic gradient for speech DNNs," in *ICASSP*, 2014, pp. 235–239.

[6] Jeffrey Dean, Greg S. Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Quoc V. Le, Mark Z. Mao, Aurelio Marc Ranzato, Andrew Senior, Paul Tucker, Ke Yang, and Andrew Y. Ng, "Large scale distributed deep networks," in *NIPS*, 2012, pp. 1223–1231.

[7] Pan Zhou, Cong Liu, Qingfeng Liu, Lirong Dai, and Hui Jiang, "A cluster-based multiple deep neural networks method for large vocabulary continuous speech recognition," in *ICASSP*, 2013, pp. 6650–6654.

[8] Alex Krizhevsky, "One weird trick for parallelizing convolutional neural networks," *CoRR*, vol. abs/1404.5997, 2014.

[9] Shiliang Zhang, Yebo Bao, Pan Zhou, Hui Jiang, and Lirong Dai, "Improving deep neural networks for LVCSR using dropout and shrinking structure," in *ICASSP*, 2014, pp. 6849–6853.

[10] Frank Seide, Hao Fu, Jasha Droppo, Gang Li, and Dong Yu, "1-bit stochastic gradient descent and its application to data-parallel distributed training of speech DNNs," in *Interspeech*, 2014, pp. 1058–1062.

[11] Martin A. Zinkevich, Markus Weimer, Alex Smola, and Lihong Li, "Parallelized stochastic gradient descent," in *NIPS*, 2010, pp. 1–8.

[12] Awni Y. Hannun, Carl Case, Jared Casper, Bryan C. Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, Adam Coates, and Andrew Y. Ng, "Deep speech: Scaling up end-to-end speech recognition," *CoRR*, 2014.

[13] Yongqiang Zou, Xing Jin, Yi Li, Zhimao Guo, Eryu Wang, and Bin Xiao, "Mariana: Tencent deep learning platform and its applications," *Proc. VLDB Endow.*, vol. 7, no. 13, pp. 1772–1777, Aug. 2014.

[14] Xie Chen, Adam Eversole, Gang Li, Dong Yu, and Frank Seide, "Pipelined back-propagation for context-dependent deep neural networks," in *INTERSPEECH*, 2012, pp. 26–29.

[15] Qingqing Zhang, Frank Soong, Yao Qian, Zhijie Yan, Jielin Pan, and Yonghong Yan, "Improved modeling for F0 generation and V/U decision in HMM-based TTS," in *ICASSP*, 2010, pp. 4606–4609.

[16] Hynek Hermansky, "Perceptual linear predictive (PLP) analysis of speech," *The Journal of the Acoustical Society of America*, vol. 87, pp. 1738–1752, 1990.

[17] John Duchi, Elad Hazan, and Yoram Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *The Journal of Machine Learning Research*, vol. 12, pp. 2121–2159, February 2011.