An Optimized Implementation of Speech Recognition Combining GPU with Deep Belief Network for IoT

Weipeng Jing[1,2(✉)], Tao Jiang[1], Mithun Mukherjee[2], Lei Shu[2,3], and Jian Kang[1]

[1] College of Information and Computer Engineering, Northeast Forestry University, Harbin, China
{weipeng.jing,taojiang920619,laurelkang}@outlook.com
[2] Guangdong Provincial Key Laboratory on Petrochemical Equipment Fault Diagnosis, Guangdong University of Petrochemical Technology, Maoming, China
{m.mukherjee,lei.shu}@ieee.org
[3] School of Engineering, University of Lincoln, Lincoln, UK

Abstract. With the advancement in Internet of Things (Iot), the speech recognition technology in mobile terminals' applications has become a new trend. Consequently, how to accelerate the training and improve the accuracy in speech recognition has attracted the attention of academia and industry. Generally, Deep Belief Network (DBN) with Graphic Processing Unit (GPU) is applied in acoustic model of speech recognition, critical research challenges are yet to be solved. It's hard for GPU to store the parameters of DBN at one time as well as GPU's shared memory is not fully used. And parameters transmission have become a bottleneck in multi-GPUs. This paper presents a new method in which the weight matrix is divided into sub-weight matrices and established a reasonable memory model. To eliminate the inefficient idle-state during data transfers, a stream process model is proposed in which the data transfer and kernel execution are performed simultaneously. Further, apply the optimized single GPU implementation to multi-GPUs and is intend to solve the parameters transmission. Experimental results show the optimized GPU implementation without violating the size limitation of GPU's memory.

Keywords: IoT · Speech recognition · DBN · GPU
Parallel computation · Mobile computing

1 Introduction

Internet of things (Iot)[1] has obtained grateful development, it has been applied in various fields. Mobile computing, as a important part of IoT, has attracted many people. Speech recognition of mobile computing is especially important for IoT. Therefore how to improve the training speed of speech recognition has

Y.-B. Lin et al. (Eds.): IoTaaS 2017, LNICST 246, pp. 251–260, 2018.
https://doi.org/10.1007/978-3-030-00410-1_30

become one of important research topics in IoT. The traditional Gauss Mixture Model and Hidden Markov Model (GMM-HMM) [2] is a typical shallow learning structure and its performance is limited under the massive data. Deep learning which was pointed by Hinton et al. [3] is useful when training massive data by "layer-by-layer initialization". Deep Belief Network (DBN), a deep neural network, consists of Restricted Boltzmann Machine (RBMs) and is widely used in speech recognition. Microsoft researchers applied the RBM and DBN into the training of speech recognition's acoustic model, and have made great success on large scale vocabulary continuous recognition task [4]. However, training complexity limits its application in the mobile computing.

Graphic Processing Unit (GPU) has advantages of higher computing density and smaller size, which has been widely used in the field of speech recognition. GPU is applied to large vocabulary continuous speech recognition in [5] and has achieved better acceleration effect compared to Multi-CPUs. CuBLAS is used for training RBM to accelerate and has achieved good results in [6]. The GPU kernels are written in [7] to accelerate the training speed of RBM, but it's too complex and the scalability is not strong. Moreover, there are a large number of parameters in RBM model.

In summary, this paper is mainly focused on the training of speech recognition based on DBN using single GPU without the limitation of DBN model's parameters and the method is applied to multi-GPUs. In addition, a suitable memory model of GPU is designed utilizing the full computing capability of GPU to accelerate the DBN training speed with low word-error-rate.

2 DBN Training

The training process of DBN model in speech recognition mainly contains layer-wise greedy unsupervised learning in pre-training and supervised learning in fine-tuning.

2.1 Pre-training of DBN

Contrastive Divergence (CD) algorithm [8] is used for unsupervised training in each layer of RBM. Each layer of RBM receives the output values of the previous layer as the input value, then the output values are propagated in RBM. Finally the output values are used as the input value of the next layer RBM. The main objective of the pre-training is to retain the characteristic information of the training speech data's feature vectors while they are mapped to the different feature spaces.

Restricted Boltzmann Machine (RBM). RBMs are the key components of the DBN with a greedy learning algorithm. A RBM is an energy-based generative model that consists of two layers: (1) a layer of binary *visible* units and (2) a layer of binary *hidden* units, with symmetrical connections. Any unit in one layer is connected to all units in the other layer and has no connection with units in the same layer.

Considering RBM's special structure, given a particular random input configuration v_i, all the hidden units h_j, are independent of each other. As a result, the probability of h_j given v_i becomes

$$P(h_j = 1|v) = sigm\left(\sum_{i=1}^{m} W_{ij}v_i + b_j\right), \tag{1}$$

where $sigm(\cdot)$ is the sigmoid function. Similarly, the probability of v_i given a specific hidden state h_j is

$$P(v_i = 1|h) = sigm\left(\sum_{j=1}^{n} w_{ij}h_j + a_i\right). \tag{2}$$

Contrastive Divergence (CD) Algorithm. Pre-training is used to adjust the trained speech data. To obtain the value of θ in RBM, a much faster method, Contrastive Divergence (CD-k) algorithm [9] is used with k-parameter as 1. According the CD-k algorithm, the update rules are used as follows:

$$\Delta W_{ij} = \varepsilon(\langle {v_i}^{(0)}, {h_j}^{(0)}\rangle, \langle {v_i}^{(k)}, {h_j}^{(k)}\rangle), \tag{3}$$

$$\Delta a_i = \varepsilon(\langle {v_i}^{(0)}, {v_i}^{(k)}\rangle), \tag{4}$$

$$\Delta b_j = \varepsilon(\langle {h_j}^{(0)}, {h_j}^{(k)}\rangle), \tag{5}$$

Taking into account the scale of the trained speech data, part of trained speech data are taken as mini-batch [7] rather than every sample of trained speech data. To reconstruct v_i using CD algorithm, it is important to consider binary hidden states.

2.2 Fine Tuning of DBN

Initial values of the entire DBN model's parameters are obtained after pre-training. The fine tuning is performed on the speech data in which each frame is labeled with a target class label. The fine-tuning process is divided into two steps as: firstly, the extracted features of speech-data from the last layer of RBM are used as input values to the BP neural network, afterward, the output values are classified by the `softmax` function; secondly, cross-entropy [10] is used as the loss function for error calculation in BP algorithm that adjusts the parameters of whole DBN model.

As BP algorithm needs a long time to update θ, SGD algorithm is used to reduce the time. After completing the speech features' classification of a mini-batch, θ is directly updated using the calculated error to accelerate the training speed in DBN.

3 GPU Implementation of Proposed Algorithm

The proposed CD-k algorithm for large-scale RBMs was implemented on Compute Unified Device Architecture (CUDA)[1] of NVIDIA GPU, which benefits graphic rendering with massive parallelism. Although GPU can achieve remarkable performance for DBN under reasonable task decomposition of RBM and memory optimization [12], advantages of GPU's memory architecture are still not fully exploited for parallel DBN. To use the GPU to accelerate the training of DBN model, the optimization of RBM's training using GPU is implemented for three aspects: based on (1) memory model with sliced weight matrix, (2) multi-streams processing model on single GPU and (3) the multi-GPUs implementation.

3.1 Memory Model Based on Sliced Weight Matrix

The training speed of RBM can be improved by training the data with size of l. However, there still exists a problem that the large size of l will hurt the overall efficiency of learning. So in the training process, the size of l chosen is much smaller than the m, n, v_i, h_j, bias a_i and bias b_j are small so that they can store in the GPU device memory. However, the weight matrix is so large due to interconnection between any two units, it is likely to occur the phenomenon the weight matrix cannot store in GPU device at one-time. Also there are also other parameters in RBM especially the some speech signals will be trained at one time rather than one. So weight matrix is divided into many sub-weight matrices $W_i \in R^{m\prime \times n}$ where $m\prime << m$ such that every W_i could be stored in the GPU device memory. The sub-weight matrices will be determined by experiments, which also means the size of $m\prime$.

The trained DBN model's parameters are copied from CPU to GPU after having divided the weight matrix into sub-weight matrices. Using (1) and (2), $h_j^{(0)}$ and $v_i^{(k)}$ are calculated. When calculating $h_j^{(0)}$ with GPU, in order to hide the global memory latency, threads are needed to use at a much finer granularity to take full use of the GPU computing resources. Hence, the connection $w_{ij} \in W_i$ is taken as the smallest unit of computation which is called thread performing a function that multiples the v_i by its weight. Every block can be represented a unit by this way. Then, $h_j^{(0)}$ is calculated.

As for the calculation of $v_i^{(k)}$, the weight matrix is divided into W_i and transferring W_i one by one on demand. However, the transfer of W_i would cost lots of time, a method is adopted that avoids the undesirable memory transfer of W_i. Because the calculation of $v_i^{(k)}$ and $h_j^{(0)}$ use the same W_i, the $v_i^{(k)}$ will be

[1] In 2006, a parallel computing platform and programming model for NVIDIA GPUs named CUDA [11] was introduced aiming to make full use of computing power of GPUs to achieve general purpose computation. CUDA also enables programmers without any knowledge about graphic APIs to write `C/C++` code for high performance scientific computation by using NVIDIA GPUs. Therefore, it is widely used in speech recognition based on DBN model.

calculated immediately after the calculation of $h_j^{(0)}$. In this way, $v_i^{(k)}$ calculation is done along with $h_j^{(0)}$ using the W_i that are already loaded and the second transfer is no longer needed, also it reduces the time of thread synchronization, accelerating the speed of RBM's training.

The same method is used to calculate $h_j^{(k)}$. After $h_j^{(k)}$ is finished, W_i is updated immediately as the updating W_i needs the corresponding $v_i^{(k)}$ and $h_j^{(k)}$ according to (3), saving another transfer of W_i for updating current W_i. Also the updating of W_i needs $h_j^{(k)}$, $h_j^{(0)}$, $v_i^{(k)}$, $v_i^{(0)}$ matrices.

3.2 Parallel Processing Based on Streams Process

The GPU implement of RBM using memory model based on sliced weight matrix has accelerated the speed. But the memory transfer becomes a major problem to reduce the training speed. For example, updating every W_i requires two memory transfers of CPU to GPU and one memory transfer of GPU to CPU. However, GPU simply keeps idling when these memory transfers occur, as a result, the GPU does not use the threads to calculate the task to be executed.

To make full use of GPU's computing resources and eliminate this inefficient idle-state, the streams process of CUDA are used to do concurrent execution of threads. There are two aspects in streams process: firstly, data transfers and kernel executions are executed concurrently in the same stream; secondly, launch multiple streams and overlapping CUDA operations in different streams to achieve concurrency execution. In our algorithm, potential concurrency exits in the two for-loops. With multiple streams launched and each responsible for a subset of W_i. The process of training RBM in opCD-k algorithm has following steps:

- *Step 1:* Copy the data from CPU to GPU,
- *Step 2:* Calculate $h_j^{(0)}$,
- *Step 3:* Calculate $v_i^{(k)}$ after k iterations of Gibbs sampling,
- *Step 4:* Calculate $h_j^{(k)}$,
- *Step 5:* Update the parameters $\theta = \{w_{ij}, a_i, b_j\}$, and
- *Step 6:* Transfer the results from GPU to CPU.

Combining the stream process of CUDA and opCD-k algorithm, it is evident that Step 1 and Step 2 could achieve concurrency execution in a stream; Step 2 and Step 3 could run concurrently as long as the computing resources on GPU allows; Step 1 and Step 6 can be performed concurrently in different streams.

3.3 Multi-GPUs Implementation

The two methods proposed above will improve the DBN's training speed. But single GPU's computation is limited. So in order to improve the speed of DBN's training, the multi-GPUs will be used. In Speech Recognition, due do the full connection of DBN model, it would cost much time to use model parallelism,

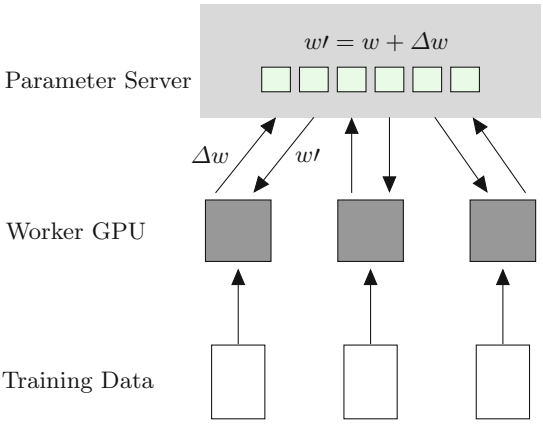

Fig. 1. Data parallelism with parameter server in multi-GPUs.

so the data parallelism is widely used. In this paper, we use parameter server to do DBN's training with multi-GPUs which are in a machine. As shown in Fig. 1, it is the data parallelism with parameter server in multi-GPUs. One GPU is chosen as parameter server. It receives one iteration's result of the parameters of DBN model from other GPUs, and it will do the update of the parameters and transfer them to the GPUs, so that the GPUs could do other iteration calculations. Under this circumstance, the bandwidth of the parameter server becomes a bottleneck.

So the total communications should be reduced. We can reduce the communication frequency by large the size of batch. But when the size of batch is too large, the data cannot store in GPU device. So we divide the batch with fetch $\times$ mini-batch. The size of mini-batch data will be copied to GPU, and copied fetch times. The result will not transfer to the parameter server instead they will be added at the local GPU. After fetch times, the final summation will be transferred to the parameter server. When deciding the size of fetch, the method of hot start is adopted. By this way, the communication frequency can be greatly reduced.

4 Experimental Results and Analysis

To verify the acceleration effect of DBN using the optimized GPU method. The input data are the 440-dimensional speech features which are spliced by 40-dimensional fMLLR features. The fMLLR features are obtained by the process of speech recognition toolkit Kaldi [13].

There are six layers of RBM in DBN model. The parameters in the two different stages of DBN model are different. The first layer of RBM is Gaussian-Bernoulli distribution, and others are Bernouli-Bernouli distributions. The Intel(R) Xeon(R) CPU E5-2620 v2 has 128G memory with a frequency of

2.10 GHZ, four GPUs which is called Tesla K20m which has 5G memory are used in the multi-GPUs. The version of operation system is Red Hat Enterprise Linux Server release 6.4, the CUDA Tookit is 7.0.

4.1 Parameters Tuning of OpCD-K Implementation

opCD-k implementation relies on weight slicing technique requiring a second transfer of W_i. In addition, stream synchronization and scheduling will introduce more overhead. All these new features of opCD-k may cause low performance. So the parameters must be set reasonable.

The Size of slice_size. The relations of the parameters in opCD-k algorithm can be described as:

$$s \times slice\_size \times n \times sizeof(float) \leq GPU\_memory \tag{6}$$

where dev_meme is GPU memory available for storing weight matrix W_i and slice_size is the rows of each W_i. We explored different combinations of these parameters. Assuming a fixed amount of device memory, which is a common case given a particular GPU, an optimal combination of s and slice size is desirable. In this experiment, we assume the device memory limits us to load only one forth of a weight matrix $W^{1024\times1024}$. Multiple configurations satisfying s $\times$ slice_size $= 1024/4$ are valid such as 3×100, 4×75 and 5×60. The chosen value is that 300 is about equal to 1024/4 for calculation. We tried a set of configurations and draw a conclusion from the result in Fig. 2 that neither large slice with few streams nor small slice with numerous streams is a good choice. From the figure we can clearly observe the training time with the configurations change. And the 4×75 is the best choice for the experiment.

4.2 RBM Training Time Evaluation

We conduct an experiment to evaluate the training time of RBM. Results of the experiment can visually reflect the performance of DBN with optimized single GPU implementation.

RBM Training Time. Experiments are conducted to compare the training time spent on one iteration of RBM with the 440 visible units and different hidden units in three different ways as follows: (1) the optimized GPU implementation with 1/4 memory usage and five streams, (2) the implementation with single GPU as in [7], and (3) the implementation of Kaldi with single GPU. Figure 3 illustrates the training time with various number of hidden units for three cases. It is observed that the cost increases with the increase of hidden units. At first, the first way spends the most time because it need more time to exchange the weight matrix and streams synchronization. however, with the increment of the number of hidden units, the first way costs less time than the others. The reason is that the first way uses reasonable memory model and streams process. The first way achieves acceleration with a maximum time of 1.7 than the third way when the number of hidden units is 2^{13}.

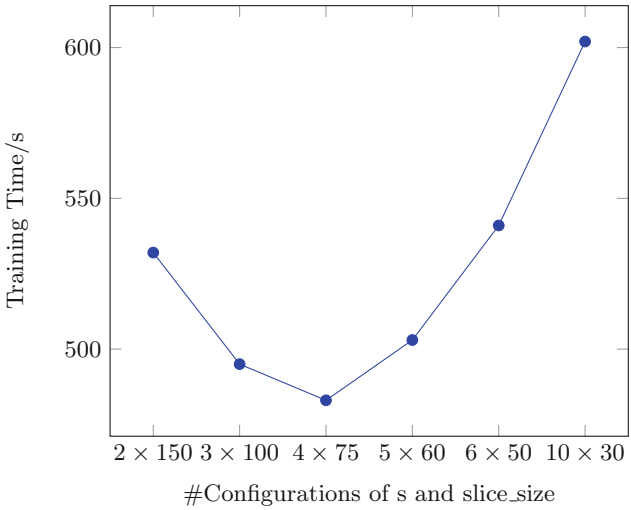

Fig. 2. Performance of different configurations of s and slice_size given $s \times slice\_size = 300$ on a RBM of 1024×1024.

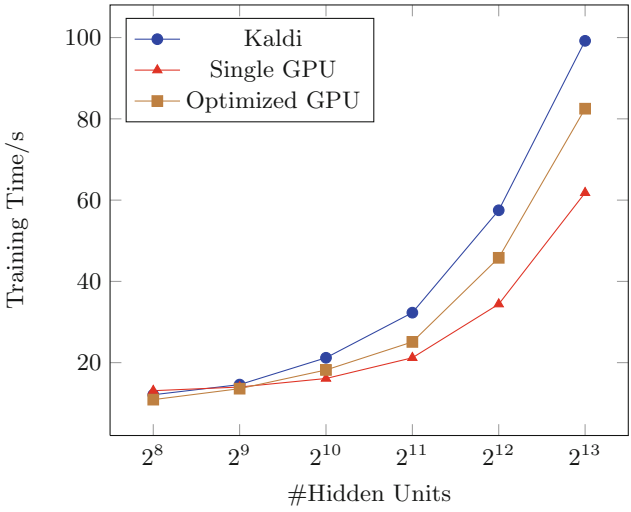

Fig. 3. RBM's training time after one iteration.

4.3 Performance of OpCD-K on DBN with Multi-GPUs

Training Time and word error rate are another two important factors to evaluate the performance of DBN. So we conduct the following experiments.

Training Time and Word Error Rate of DBN with Multi-GPUs. DBN training time is evaluated in four different ways listed in Table 1. It is observed that

Table 1. DBN'S training time with multi-GPUs

Model	Time/hour
Kaldi with single CPU	2.5
Kaldi with multi-GPUs	0.728
Multi-GPUs implementation in [14]	0.65
Optimized GPU implementation with multi-GPUs	0.54

the multi-GPUs implementation has greatly reduced the time of training DBN, which the time cost is only 0.52 h. It has achieved about 1.35 times and 1.2 times comparing with the Kali implementation and GPU implementation in [14]. Also it obtains the acceleration of 4.6 times than using the Kaldi with single GPU. Therefore, the optimized GPU implementation with multi-GPUs makes a great acceleration on training DBN model.

In the experimental part of the word error rate. It is observed from Table 2 that the optimized GPU implementation only has a 7% and 5% performance loss comparing with Kaldi with single GPU and multi-GPUs respectively.

Table 2. Training DBN's word error rate with multi-GPUs

Model	Error rate
Kaldi with single GPU	18.8%
Kaldi with multi-GPUs	19.3%
Multi-GPUs implementation in [14]	23.1%
Optimized GPU implementation with multi-GPUs	20.2%

5 Conclusion

This paper presents an efficient parallel algorithm to overcome the problem of huge parameters and unreasonable usage of GPU's memory model while accelerating the computation speed of speech recognition in wireless networks based on DBN. Aiming to fully utilize GPU's computing resources, we divide the weight matrix into sub-weight matrices to parallel the RBM's training for large parameters. In order to optimize the training of RBM, we adopt multi-streams processing model for single GPU. In addition, to cope with the limitations of a single GPU computing capability, We extend from single GPU to multi-GPUs to achieve better results. The future work includes to parallelize the computation of DBN on GPU-accelerated cluster to model large-scale problems under noisy environment.

Acknowledgement. The work described in this paper is supported by Guangdong Provincial Key Laboratory of Petrochemical Equipment Fault Diagnosis, Guangdong University of Petrochemical Technology (GDUPTKLAB201502) and Special Fund for Forest Scientific Research in the Public Welfare (201504307).

References

1. Gubbi, J., Buyya, R., Marusic, S., Palaniswami, M.: Internet of things (IoT): a vision, architectural elements, and future directions. Future Gen. Comput. Syst. **29**(7), 1645–1660 (2013)
2. Su, D., Wu, X., Xu, L.: GMM-HMM acoustic model training by a two level procedure with Gaussian components determined by automatic model selection. In: IEEE International Conference on Acoustics Speech and Signal Processing, pp. 4890–4893, Texas, USA, March 2010
3. Hinton, G.E., Salakhutdinov, R.R.: Reducing the dimensionality of data with neural networks. Science **313**(5786), 504–507 (2006)
4. Seide, F., Li, G., Yu, D.: Conversational speech transcription using context-dependent deep neural networks. In: Conference of the International Speech Communication Association (INTERSPEECH), Florence, Italy, pp. 437–440, August 2011
5. Sainath, T.N., Kingsbury, B., Ramabhadran, B., Fousek, P.: Making deep belief networks effective for large vocabulary continuous speech recognition. In: Proceedings of Automatic Speech Recognition and Understanding, pp. 30–35, December 2011
6. Raina, R., Madhavan, A., Ng, A.Y.: Large-scale deep unsupervised learning using graphics processors. In: Proceedings of International Conference on Machine Learning (ICML), Montreal, Quebec, Canada, pp. 873–880, June 2009
7. Lopes, N., Ribeiro, B.: Towards adaptive learning with improved convergence of deep belief networks on graphics processing units. Pattern Recogn. **47**(1), 114–127 (2014)
8. Hinton, G.E.: Training products of experts by minimizing contrastive divergence. Neural Comput. **14**(8), 1771–1800 (2002)
9. Swersky, K., Chen, B., Marlin, B., De Freitas, N.: A tutorial on stochastic approximation algorithms for training restricted Boltzmann machines and deep belief nets. In: Information Theory and Applications Workshop, pp. 1–10, January 2010
10. Deng, L., Togneri, R.: Deep dynamic models for learning hidden representations of speech features. In: Ogunfunmi, T., Togneri, R., Narasimha, M. (eds.) Speech & Audio Processing for Coding Enhancement & Recognition, pp. 153–195. Springer, Heidelberg (2015). https://doi.org/10.1007/978-1-4939-1456-2_6
11. NVIDIA. What is CUDA (2006)
12. Wang, Y., Tang, P., An, H., Liu, Z., Wang, K., Zhou, Y.: Optimization and analysis of parallel back propagation neural network on GPU Using CUDA. In: Arik, S., Huang, T., Lai, W.K., Liu, Q. (eds.) ICONIP 2015. LNCS, vol. 9491, pp. 156–163. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-26555-1_18
13. Povey, D., et al.: The Kaldi speech recognition toolkit. IDIAP Publications (2012)
14. Xue, S., Yan, S., Dai, L.: Fast training algorithm for deep neural network using multiple GPUs. J. Tsinghua Univ. (Sci. Technol.) **53**(6), 745–748 (2013)