A Practical Implementation of GPU based Accelerator for Deep Neural Networks

Jian Ouyang[1], Lei Jia[1], Zhenyu Hou[1], Guangyu Sun[2], Guangjun Xie[1] ,Yong Wang[1]
[1]Baidu, Inc.
[2]Peking University
[1]{ouyangjian, jialei, houzhenyu, xieguangjun,wangyong03}@baidu.com
[2]{gsun}@pku.edu.cn

Abstract:

Deep neural network (DNN) has been widely used in Internet applications like speech recognition, image classification, neural language processing, etc.. DNN training process, however, is very computing intensive and is also difficult to be handled by distributed clusters without using algorithm approximation [6], which incurs recognition accuracy loss. As training data size increases for modern applications, e.g. large vocabulary conversational speech recognition, the training process becomes more and more time consuming. In order to accelerate the process, we implement a GPU based DNN accelerator in a X86 computing system, which is. The original DNN training algorithm is implemented on GPU without approximation. Our heterogeneous architecture with a single GPU accelerator can achieve 462X speedup compared to the homogeneous architecture with only one X86 CPU. In addition, we implement a heterogeneous system using multiple accelerators, which further gains 1.99X and 3.3X performance improvements with two and four GPUs, respectively.

1. Introduction:

Recently, speech recognition application is becoming one of killer APPs for mobile network. Many internet companies have released their speech recognition APPs for mobile devices, e.g. Google's "Google now" and Apple's "Siri".

As the leading search engine in China, Baidu also provides voice services, such as voice search, voice input on mobile devices. In these recognition APPs, deep neural networks (DNN) has been widely adopted as a promising acoustic-modeling technique [4]. Compared to traditional GMM/HMM based algorithm, DNN can achieve a significant higher accuracy for speech recognition [2][3].

On the other hand, the DNN training is process is really computing intensive. For example, it takes about 3 months to train a 24-hour speech data set on a traditional CPU based homogeneous system [1]. As speech data size increases to thousands of hours, which is common for modern speech recognition APPs, training time on a traditional system becomes unacceptable [2]. Recently, GPU based DNN training accelerator has been proposed to accelerate the process [4]. The prior work, however, employs an approximation of DNN training algorithm, which incurs data accuracy loss. Moreover, further accuracy loss is induced with a large input batch size in their design.

In this work, we propose a practical implementation of GPU based accelerator for DNN, which has already been adopted in Baidu's commercial speech recognition APPs. Compared to prior work, our contributions are summarized as follows,

- An original DNN algorithm without approximation is implemented in both single and multiple GPU based accelerators with proper optimization techniques.
- A small batch size with 100 is used to improve recognition accuracy.
- Using our heterogeneous platform, the performance of DNN training process can be significantly improved compared to the homogeneous one.
- We compared our implementation with prior approach and achieve similar or even better performance with higher data accuracy.

The rest of this paper is organized as follows. In the next section, we present a brief review of DNN structure and quantitatively evaluate computing complexity of using DNN for speech recognition. In Section 3, we introduce our accelerator implementation using single and multiple GPUs. At the same time, comparisons to homogeneous system and prior heterogeneous approach are also provided. The paper is concluded in Section 4.

2. DNN Structure Review

Figure 1 demonstrates the structure of DNN. The number of middle layer of a real speech recognition system is typical more than five, and the number of nodes of middle layer is typically set to 2048. The input vector describes the feature of training voice. Every input vector contains 429 elements and one second of speech data is composed of 100 vectors. The output layer typically contains 9000 nodes.

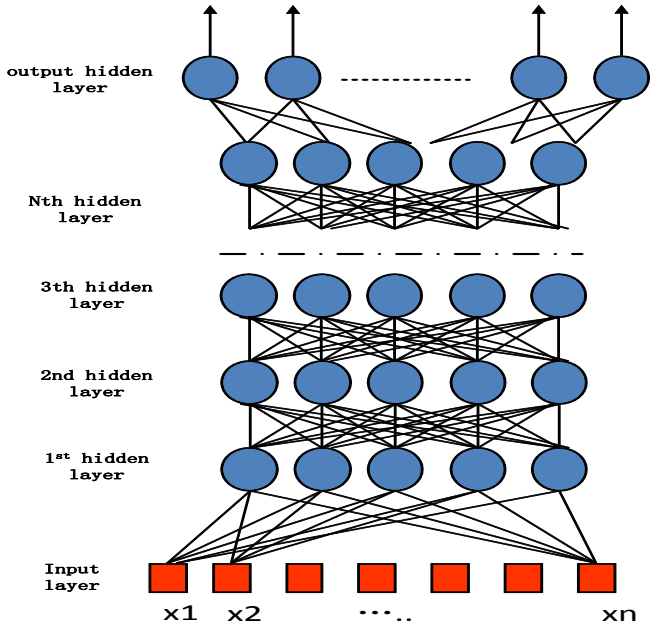

Figure 1: structure of DNN

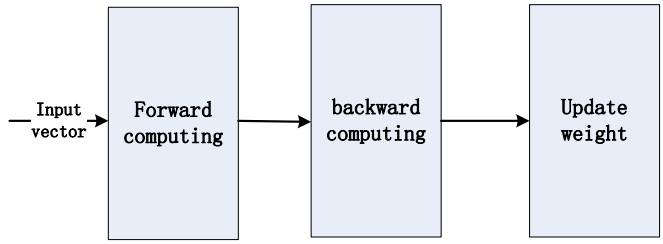

Figure 2: processing of DNN training

The calculation step of a real DNN training system for speech recognition is illustrated in Figure 2. The calculation is sequentially processed step by step. Within each step, data is processed through different layers. In a forward computing step, the process starts from the first layer and each following layer takes output of its previous layer as input. The computing ends when it reaches the last output layer of the set.

On the contrary, a backward calculation is a reverse process of a forward one. It will calculate the error vector of the last output layer and go through all layers to the first one. The update weight process is similar as forward computing. It updates the weight matrix of each layer one by one till the last layer. It is difficult to distribute calculation to several servers because each calculating of layers is absolutely sequential and the weight matrix of each layer must be synchronized for each input vector. Every input vector would be calculated in such three states. One round process of calculating all input vectors is called an epoch. A real system would often run 20 to 50 epochs to train a result with enough accuracy.

In a real system, a batch of 100 to 1024 input vectors is computed simultaneously to increase parallelism. However a larger batch size means lower accuracy [5]. The initial size of a batch would be 100 and increase to 200 or 400 during the later epochs. The batch size, however, cannot exceed1000, otherwise the accuracy would degrade significantly. The typical computing operation of each layer is matrix multiplication, matrix addition and log for each elements of matrix.

The calculation intensity of each batch during one epoch is estimated in Table 1. The configuration details are as follows. There are 7 layers. Batch size is set to 100. The output nodes and middle nodes are 8000 and 2048, respectively. The input vector is 429.

Forward	4.0G float mul-add ops
Backward	2.6G float mul-add ops
Update weight	4.4G float mul-add ops

Table 1: Calculation complexity of a 7 layer DNN

If the multiply-and-add instruction is enabled in a 2.4GHz i7-like CPU, it takes about 3.3 seconds to process a second of training data without considering the cache miss and CPU core pipeline exceptions. A typical 1000 hours data would cost several months by traditional CPU. Actually, a real training would include two individual steps: NO-Sparse training with dense weight matrix and sparse training with sparse weight matrix. So the intensity of calculation in a real system would be increased to twice of the estimation. Since DNN training is composed of various matrix calculations, it is suitable to accelerate such computing by GPU. The latest GPU like Nvidia C2050 is able to achieve 300Gflops for double precision matrix multiplication and 600Gflops for float precision matrix multiplication while batch size is 200. It is at least 50 times more powerful than a six core X86 CPU.

3. Accelerator Implementation

a) Single GPU Implementation

Table 2 lists our experiment environment.

CPU	Intel E5620x2 2.4GHz
GPU	Nvidia C2050

Host Memory	64GB
GPU memory	2.6GB
Software	RH Linux 6.0/GCC 4.6/ CUDA 4.1

Table 2: experiment environment

The detailed configuration of heterogeneous system in our implementation is listed in Table 2. The whole DNN is composed of eight layers. Its output layer has 9000 nodes and the middle layer has 2048 nodes. The input vector has 429 elements and the batch size is set to 100. Table 3 compares the measured timing consumption of processing one batch using homogeneous and our heterogeneous systems.

	CPU Only (ms)	CPU+ GPU (ms)	Speed up
Forward	10473	19	551
Backward	14890	20	744
Update weight	3418	21	162
total	26985	61	442

Table 3: Comparison of timing consumption.

The real training system of Baidu uses more than 8 layers and the size of output layer is 9000. And each iterates more than 16 epochs. We trained 50 hours data in our real training system with the batch size of 100 and got a 462 times faster than a signal CPU core.

b) Multiple GPU implementation

With a single GPU, it would cost nearly 4 months to train 2000 hours data. The major challenge in real product is to find a scalable solution to reduce training time as training data size increases. We have mentioned that DNN algorithm is difficult to be distributed into multiple servers for parallel computing because the weight matrix of each layer must be synchronized after updating. The network overhead would degrade the overall performance significantly. Thus, we further implement the training accelerator in one server using multiple GPUs.

A naive method for multiple GPUs is shown in Figure 4.

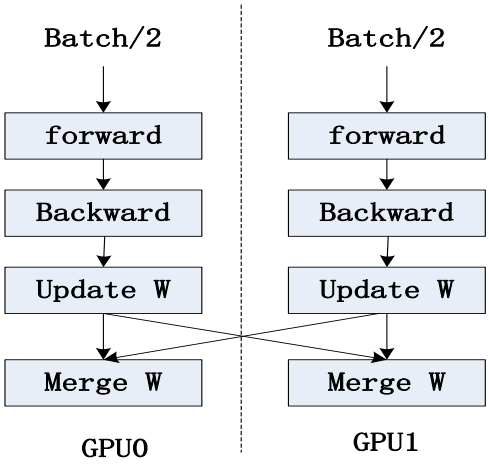

Figure 4: A naive method for multiple GPUs

The naive method is composed of two parts. Each GPU perform a process individually as a single GPU implementation. At last, two GPUs fetch the weight matrix of every layer from each other and merge them into an integrated weight matrix. This naive method would lead to heavy communication overhead. A DNN with 7 layers would need 50ms to synchronize the weight matrix between 2 GPUs for a batch with size 100. The communication time is even more than the entire computing time by a signal GPU implementation. Thus, we propose another improved method, as described in Figure 5.

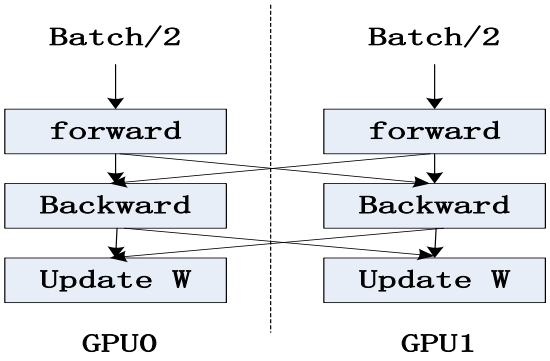

Figure 5: an improved strategy

This method synchronizes output of forward step and backward step for each layer because the output of forward step and backward step is 100 times smaller than weight matrix. Consequently, the total communication time is only 1.7ms. Since the outputs of preceding step have been synchronized. The merge W step is not required in this method. Another optimization of this method is to pipeline the computing and communication so that the time of communication is hidden by computing. The pipeline design is illustrated by Figure 6.

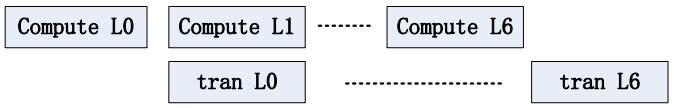

Figure 6: Structure of pipeline design.

We achieved 1.99x speedup by this method compared to a single GPU implementation with the batch size of 100.

We also propose another more powerful architecture using four GPUs. We found that if the entire computing task is distributed to multiple GPUs uniformly, the communication overhead would be significantly increased. The overall performance would be degraded seriously. Unlike the two-GPU design, we propose an asymmetric architecture for four GPUs. This architecture is demonstrated in Figure 7.

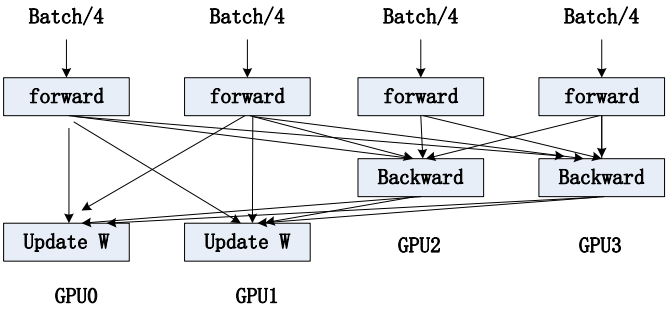

Figure 7: 4 GPUs architecture

The update Weight step in GPU0 and GPU1 is simultaneously processed with the backward step in GPU2 and GPU3. For example, the completion of backward of layer 6 would trigger update W of layer 6, and the update W of layer 6 is calculated in parallel with backward of layer 5. We achieve another 3.3X speedup using this four-GPU architecture compared to a single-GPU accelerator with the batch size of 100.

Compared to prior approach, our two-GPU solution is able to achieve better performance Moreover, algorithm approximation and large batch size is not required. And our 4 GPUs architecture is easy to scale up to 8 GPUs.

Conclusion

In this paper, we implement DNN training accelerators using GPU on Baidu's real system with real data set. We have shown that 462 times speedup is achieved by single GPU compared to latest homogeneous X86 CPU architecture. And we also propose techniques to implement DNN algorithms on multiple GPUs. Compared to prior approach, these methods can get nearly linear speedup without using approximant algorithm and large batch size, which result in loss of recognition accuracy.

[1] George E.Dahl, Dong Yu, Li Deng, Alex Acero, "Context-Dependent Pre-trained Deep Neural Networks for Large Vocabulary Speech Recognition," IEEE Trans. Speech and Audio Pro, Special Issue on Deep Learning for Speech and Lang. Processing, 2012

[2] Frank Seide, Gang Li, Xie Chen, Dong Yu, "Feature Engineering in Context-Dependent Deep Neural Networks for Conversational Speech Transcriptioon," Automatic Speech Recognition and Understanding Workshop, 2011

[3] Navdeep Jaitly, Patrick Nguyen, Andrew Senior, Vincent Vanhoucke, "Application of Pretrained Deep Neural Networks to Large Vocabulary Conversational Speech Recognition", http://learning.cs.toronto.edu, 2012

[4] Xie Chen, Adam Eversole, Gang Li, Dong Yu, Frank Seide, "Pipelined Back-Propagation for Context-Dependent Deep Neural Networks", 13th Annual Conference of the International Speech Communication Association, 2012

[5] Frank Seide, Gang Li, Dong Yu, "Conversational Speech Transcription Using Context-Dependent Deep Neural Networks" International Speech Communication Association, 2011

[6] Alain Petrowski et al., "Performance Analysis of a Pipelined Back propagation parallel Algorithm ", IEEE Trans, Neural Networks, 1993