ACCELERATING MULTI-USER LARGE VOCABULARY CONTINUOUS SPEECH RECOGNITION ON HETEROGENEOUS CPU-GPU PLATFORMS

Jungsuk Kim and Ian Lane

Carnegie Mellon University
Electrical and Computer Engineering
5000 Forbes Avenue, Pittsburgh, PA 15213-3890, USA
jungsuk@cmu.edu, lane@cmu.edu

ABSTRACT

In our previous work, we developed a GPU-accelerated speech recognition engine optimized for faster than real time speech recognition on a heterogeneous CPU-GPU architecture. In this work, we focused on developing a scalable server-client architecture specifically optimized to simultaneously decode multiple users in real-time. In order to efficiently support real-time speech recognition for multiple users, a "*producer/consumer*" design pattern was applied to decouple speech processes that run at different rates in order to handle multiple processes at the same time. Furthermore, we divided the speech recognition process into multiple consumers in order to maximize hardware utilization. As a result, our platform architecture was able to process more than 45 real-time audio streams with an average latency of less than 0.3 seconds using one-million-word vocabulary language models.

Index Terms— Automatic Speech Recognition, Graphics Processing Unit, Distributed Speech Recognition, Multi-user speech recognition

1. INTRODUCTION

Modern multi-user applications are often challenged by the need to scale to a potentially large number of users while minimizing the degradation in service response even under peak load conditions. Large vocabulary continuous speech recognition (LVCSR) applications present an additional hurdle because of the disparity between the number of potentially active users and the limited system ability to provide computationally intensive automatic speech recognition (ASR) services [1].

Traditional ASR engine implementation in distributed speech recognition (DSR) system assigns one thread or a process per client until the number of clients approaches the server peak capacity to prevent performance degradation [1, 2, 3]. This approach generally works well on modern homogeneous multi-core CPU platforms. The GPU-accelerated speech recognition engine is a good alternative solution to overcome the capacity limitation of conventional multi-core ASR systems because GPUs can accelerate decoding speed significantly [4]. However, GPUs can only process one GPU kernel at a time and the number of GPU devices per server are limited. GPU processes can become a serial bottleneck in the overall ASR system although the GPUs significantly accelerate computationally intensive phases. Therefore, a multi-user ASR engine architecture needs to be specially optimized for CPU-GPU heterogeneous platform to efficiently support many users.

When optimizing a multi-user speech recognition engine on the CPU-GPU heterogeneous platform, there are two critical factors that must be considered. The first factor is the optimization of memory usage. Memory must be efficiently managed in order to simultaneously support hundreds of users per server while retaining large vocabularies and managing complex acoustic and language models. The second critical factor is maximizing hardware utilization. In real-time audio streams only allow a limited number of audio samples. That is insufficient to fully utilize GPU computation capability.

In this paper, we will extend our previous work, [4], to handle multiple users more efficiently on the CPU-GPU heterogeneous platform. We will introduce a "*producer/consumer*" design pattern to share large models efficiently and we will also propose a new ASR architecture to maximize the utilization of GPU by interleaving audio frames from multiple clients.

2. PREVIOUS WORKS

In order to handle as many clients as possible, an ASR should decode quickly. [5, 6, 7, 8, 9, 10] improved decoding speeds significantly by implementing an ASR engine on a GPU. In [11, 4], we proposed a novel on-the-fly rescoring algorithm specifically optimized for heterogeneous CPU-GPU platforms to resolve GPU memory limitations when large language models are used ($\geq$ 12GB). In this approach, acoustic score computation and n-best Viterbi search were performed on the GPU using a WFST composed with a small language model. During the graph search, partial hypotheses were copied and rescored on-the-fly using a large language model stored on the CPU. Using this approach, we were able to realize recognition with a 1 million vocabulary language model at 11$\times$ faster than real-time, and approximately 22$\times$ faster than a highly optimized single-threaded CPU implementation. However, extending this work to support multiple users is challenging.

Previous research, [1, 2, 3, 12], has proposed server arrangements to improve the capacity and efficiency of the overall speech recognition system. [1] suggest an event-driven, input-output non-blocking server framework, where the dispatcher, routing all the systems events, buffers the clients queries on the decoder proxy server, which redirects the requests to the one of free ASR engines. [12] presents an alternative architecture, where the entire ASR system has been decomposed into 11 functional blocks and interconnected via hub to allow a more efficient parallel use of the ASR system. However, these works investigate only the optimal server arrangement assuming a ASR engine can support multiple users efficiently. [13] proposed a GPU-accelreated ASR engine architecture investigating an optimal task scheduling to minimize the task wait time and share

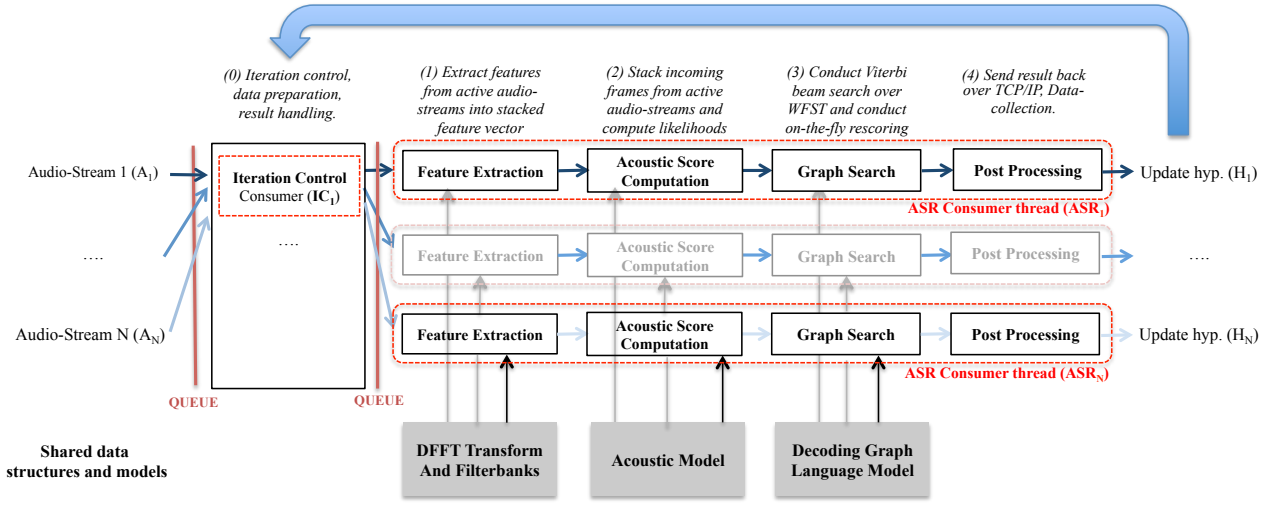

Fig. 1. Baseline multi-user speech recognition engine system architecture

the acoustic model parameters to process more users. However, this work did not propose an ASR engine architecture that leveraged mutli-core CPUs and GPUs at the same time. The evaluation was also limited because it used simulated audio traces.

3. GPU-ACCELERATED MULTI-USER SPEECH RECOGNITION

Executing multiple single user ASR engine processes with a simple router process to assign a client to each ASR is a straightforward approach to developing a multi-user ASR engine using an existing ASR engine. In this approach, each ASR process could be idle because the real-time audio stream allows only a limited number of audio samples in a given time. a greater number of ASR processes have to be initiated than the number of processors to achieve maximum hardware utilization. This may cause severe context switching and synchronization overheads that degrade system responsiveness. In addition, there could be possible redundant memory allocations if each ASR process loads large models (i.e. language model) separately.

To resolve these issues, we introduce the "*producer/consumer*" designed pattern to both the baseline and the proposed multi-user ASR engine architecture. The "*producer/consumer*" design pattern is based on the "*master/slave*" pattern, and its objective is to enhance data sharing between multiple loops running at different rates. As with the standard "*master/slave*" design patterns, the "*producer/consumer*" pattern is used to decouple processes that produce and consume data at different rates. Data queues are used to communicate data between loops. These queues offer the advantage of data buffering between producer and consumer loops. The "*producer/consumer*" pattern has the ability to easily handle multiple processes at the same time while iterating at individual rates. We completely decouple intermediate decoding results (i.e. partial hypotheses, lattice, etc.) from the ASR engine and store it in task independently. In this way, the consumer threads can share and process multiple tasks in time-multiplexed manner.

3.1. Baseline System Architecture

We developed a baseline ASR engine architecture applying the "*producer/consumer*" design pattern to the straightforward approach as shown in Fig. 1. In this architecture, the consumer loop consists of two types of consumers; an Iteration control consumer (C_{IC}), and an ASR consumer (C_{ASR}). These consumers are connected each other by two non-blocking queues; Q_{IC}, Q_{ASR}. Note that each consumer type can have multiple consumer threads.

First, the producer thread (main thread) accepts an incoming client request, over a communication channel (i.e. TCP/IP, UDP), creates a task and adds it to Q_{IC}. Then, any available C_{IC} thread takes this task from the queue, initializes data structures and adds it to next queue, Q_{ASR}. In order to maximize processor utilization, n_f (i.e. 32 frames, 320 ms) speech frames of the task will be interleaved. Note that the task is added to the Q_{ASR} only when enough speech samples are received. In ASR consumer threads extract n_f feature vectors and compute acoustic scores. These acoustic scores will be used to conduct the n-best Viterbi beam search over the WFST network composed with small language model. Partial hypotheses generated during the search are rescored using the large language model as explained in [4]. All consumer threads share an acoustic model, a WFST network and a large language model to prevent redundant memory allocations. In the end, the partial recognition result will be sent to the client if it is updated and the task will be added to Q_{IC} until the end of the audio stream.

3.2. Proposed System Architecture

Usually, a GPU works like a co-processor in CPU-GPU heterogeneous platform. This means only one task can be processed on a GPU at a time. For example, Compute Unified Device Architecture (CUDA) enabled GPUs have thousands of CUDA cores and can initiate more than thousands of threads concurrently. However, these threads execute one instruction at a time following the Single Instruction Multi Thread (SIMT) scheme similar to the vector processor [14]. Some GPU devices can execute multiple kernels concurrently. In most cases, however, only a single GPU kernel can be processed at a time because each GPU kernel uses the full computational capability of a GPU. Therefore, GPU kernels will be serialized

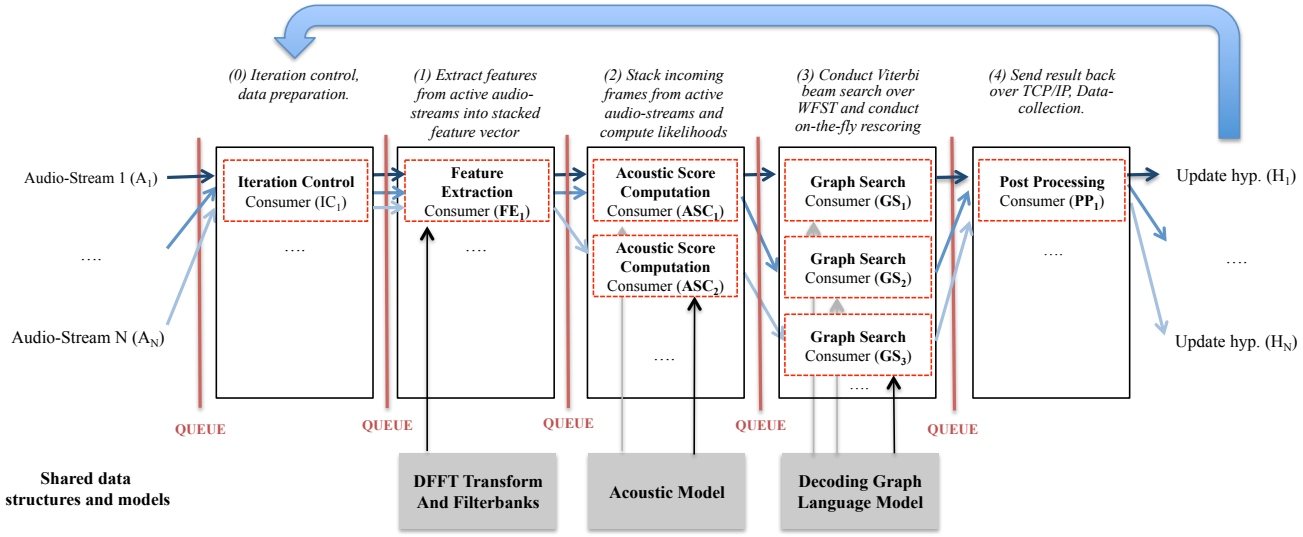

Fig. 2. Proposed multi-user speech recognition engine system architecture

when multiple CPU threads are sharing a GPU. As a result, the ASR consumer threads in the baseline architecture are not able to process assigned tasks independently. This severely limits the advantage of the "producer/consumer" design pattern.

To resolve this issue, the proposed architecture divides the ASR consumer into five different types of consumers as shown in Fig. 2; Iteration control consumer (C_{IC}), Feature extraction consumer (C_{FE}), Acoustic score computation consumer (C_{ASC}), Graph search consumer (C_{GS}), and the Post processing consumer (C_{PP}). These consumer threads are cascaded via five queues; Q_{IC}, Q_{FE}, Q_{ASC}, Q_{GS}, and Q_{PP}. In this architecture, each consumer can collect multiple tasks in the queue to improve GPU utilization because GPU computes larger data more efficiently than multiple executions of smaller data sets [14]. Therefore, the acoustic score computation consumer can process more data by interleaving audio frames from multiple tasks. In addition, this architecture allows assignment of more CPU threads or GPUs to the bottleneck consumer type to improve the overall ASR engine performance. Furthermore, updating ASR algorithm is more convenient than the baseline architecture. For example, the "on-the-fly rescoring" algorithm can be replaced with a lattice rescoring algorithm by replacing the graph search consumer without modifying other consumers. However, the proposed architecture has a complex thread configuration. The number of threads per each consumer type should be selected carefully in order to achieve maximum throughput without latency degradation.

4. EXPERIMENTAL EVALUATION

We evaluated the effectiveness of the proposed multi-user speech recognition engine architecture on a CPU-GPU heterogeneous platform on a large vocabulary version of the WSJ task as explained in [4]. A deep neural network (DNN) acoustic model was trained using the WSJ data set with 23 dimensional Filterbank coefficient feature with with global cepstral mean normalization as described in [15, 16]. The resulting acoustic model contained 3,431 context dependent phonetic states and consisted of 5 hidden layers, each with 2,048 hidden units.

A WFST was composed with a highly pruned 1 Million vocabulary 3-gram language model as shown in Table 1. The composed WFST is optimized offline for efficient parallel time-synchronous graph traversal on a GPU-accelerated platform as described in [8, 9]. A 4-gram language model was applied during decoding for partial hypotheses rescoring. The evaluation platform consisted of a single NVIDIA Titan X GPU processor with an Intel Xeon 14-core CPU as shown in Table 2.

n-gram	# of n-gram	WFST (MB)	LM binary (MB)
3 (Pruned)	6.3M	1,258	173
4	769.9M	-	19,543

Table 1. Size of WFSTs and binary language models with 1M Vocabulary.

	CPUs	GPUs
O.S.	Ubuntu 14.04 LTS	
Processor	Xeon E5-2697v3	NVIDIA Titan X
Cores	14 physical cores	3072 CUDA Cores
Clock speed	2.60 GHz	1.22GHz
Memory	128 GBytes DDR4	12 GBytes GDDR5

Table 2. Evaluation Platform Specification.

Using these models, the best WER we achieved was 5.1% with a decoding speed of 0.07 RTF, in the offline decoding evaluation. We selected our operating point at WER 5.33% where decoding speed was 0.02 RTF. During the online evaluation, the real-time audio streaming clients read t seconds of audio samples from the audio file and send it as an audio message in t seconds interval. In this evaluation 2,048 audio samples (16KHz, 16 bits/sample) are sent to the ASR server for every audio message transaction. Clients are located in separated servers and are connected with the ASR server via the wired local network. These audio messages are collected until 32 batch audio frames (320 ms) are interleaved for batch processing. The number of frames per batch is closely related to the partial recognition result update frequency and the end of speech detection. Therefore, one should not use very large size batches in a real-time audio transcription tasks.

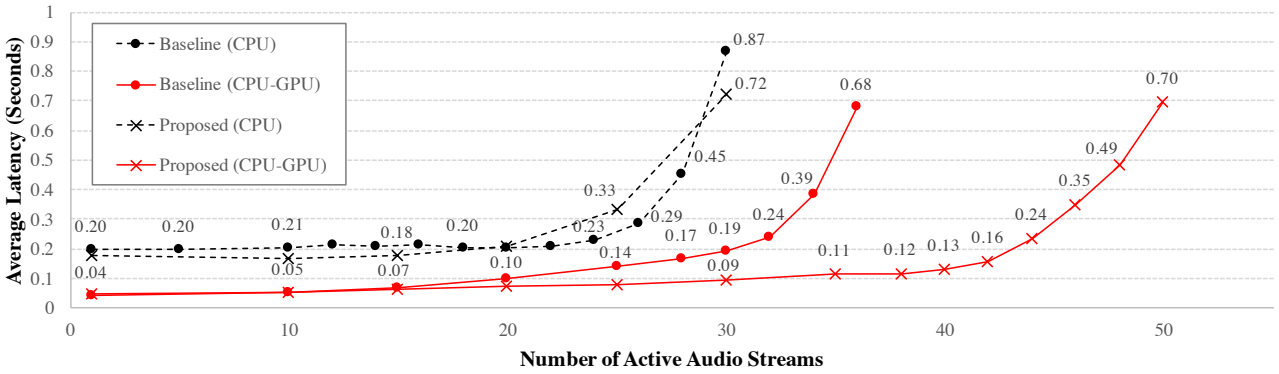

Fig. 3. Relationship between number of the active real-time audio streams and the average latency.

	CPU		CPU-GPU	
	Baseline	Proposed	Baseline	Proposed
# lookup	1		8	
# C_{IC}	2	1	2	1
# C_{ASR}	14×1	-	2×8	-
# C_{FE}	-	2	-	2
# C_{ASC}	-	10	-	1
# C_{GS}	-	4×1	-	2×8
# C_{PP}	-	2	-	1
Total	16	19	18	22

Table 3. Consumer threads configurations (C_{ASR} and C_{GS} use '# lookup' threads per consumer for the parallel language model lookups) [4].

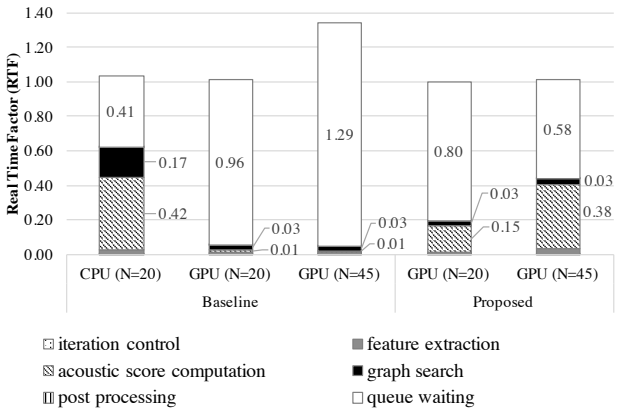

Fig. 4. Processing speed (RTF) of ASR phases (N = number of active audio streams).

Table. 3 shows thread configurations. As we mentioned in Section 3.2, in order to get the maximum throughput while maintaining low latency, the number of threads for type consumer type should be selected carefully. Thread configurations in Table 3. were selected carefully after comparing many different possible configurations.

4.1. Decoding Performance

We compared baseline architecture and proposed architecture on both homogeneous multi-core CPU platform and CPU-GPU heterogeneous platform as described in Table 3. For the ideal user experience, ASR server should have capability to provide results in a given time (latency) regardless of speech duration. In this evaluation, we measured average latency between the end of speech in the audio stream to the arrival of the final recognition result when multiple active audio streams are processed concurrently to evaluate responsiveness of multi-user speech recognition systems.

Figure 3 shows the relationship between the average latency and the number of active clients. The proposed architecture of the CPU-GPU heterogeneous platform, "*Proposed (CPU-GPU)*", handled more than 45 active real-time audio streams at an average latency of 0.3 seconds which is 1.36 times more server capacity compared to the GPU baseline architecture, "*Baseline (CPU-GPU)*", and 1.73 times more than the CPU baseline architecture, "*Baseline (CPU)*". In addition, proposed architecture improved the average latency from 0.2 sec to 0.04 sec when processing single client by accelerating the acoustic score computation and the graph search phases 42× and 5.6× respectively.

Fig. 4 shows the processing time (RTF) of all ASR sub processes when the number of active clients (N) are 20 and 45. As we explained in Section 3.1, the baseline architecture processes a single task at a time on the GPU. Therefore, speeds of ASR sub-processes are unaffected when the ASR system handled more active clients. The waiting times were increased because the GPU was fully occupied. Acoustic score computation time increased only 15 times and 38 times when processing 20 and 45 users in the proposed architecture because it utilizes maximum GPU computational capability by interleaving audio frames from multiple tasks as explained in Section 3.2.

5. CONCLUSIONS

In this paper, we proposed a novel multi-user speech recognition architecture on a heterogeneous CPU-GPU platform. Proposed architecture efficiently shared large language models and acoustic models by applying the "*producer/consumer*" design pattern and maximizing GPU utilization and computational capability by interleaving audio frames across the different users. As a result, the proposed architecture was able to process more than 45 concurrent real-time audio streams in less than 0.3 seconds. The architecture also able to process 1.73 times more real-time audio stream than the CPU baseline architecture and 1.36 times more clients than the GPU baseline architecture.

6. REFERENCES

[1] R.C. Rose, I. Arizmendi, and S. Parthasarathy, "An Efficient Framework for Robust Mobile Speech Recognition Services," in *Acoustics, Speech, and Signal Processing, 2003. Proceedings. (ICASSP '03). 2003 IEEE International Conference on*, April 2003, vol. 1, pp. I–316–I–319 vol.1.

[2] W. Zhang, L. He, Y.-L. Chow, R. Yang, and Y. Su, "The Study on Distributed Speech Recognition System," in *Acoustics, Speech, and Signal Processing, 2000. ICASSP '00. Proceedings. 2000 IEEE International Conference on*, 2000, vol. 3, pp. 1431–1434 vol.3.

[3] Y.-S. Chang, S.-H. Hung, N.J.C. Wang, and B.-S. Lin, "CSR: A Cloud-Assisted Speech Recognition Service for Personal Mobile Device," in *Parallel Processing (ICPP), 2011 International Conference on*, Sept 2011, pp. 305–314.

[4] J. Kim and I. Lane, "Accelerating Large Vocabulary Continuous Speech Recognition on Heterogeneous CPU-GPU Platforms," in *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, May 2014, pp. 3291–3295.

[5] P. R. Dixon, T. Oonishi, and S. Furui, "Fast Acoustic Computations using Graphics Processors," in *Proc. ICASSP*, Apr. 2009.

[6] P. Květoň and M. Novák, "Accelerating Hierarchical Acoustic Likelihood Computation on Graphics Processors," in *Proc. Interspeech*, Sep. 2010.

[7] P. R. Dixon, T. Oonishi, and S. Furui, "Harnessing graphics processors for the fast computation of acoustic likelihoods in speech recognition," *Computer Speech & Language*, vol. 23, no. 4, pp. 510–526, 2009.

[8] J. Chong, E. Gonina, Y. Yi, and K. Keutzer, "A Fully Data Parallel WFST-based Large Vocabulary Continuous Speech Recognition on a Graphics Processing Unit," in *Proc. Interspeech*, Sep. 2009, pp. 1183–1186.

[9] J. Kim, K. You, and W. Sung, "H- and C-level WFST-based Large Vocabulary Continuous Speech Recognition on Graphics Processing Units," in *Proc. ICASSP*, May 2011, pp. 1733–1736.

[10] K. You, J. Chong, Y. Yi, E. Gonina, C. J. Hughes, Y.-K. Chen, W. Sung, and K. Keutzer, "Parallel Scalability in Speech Recognition," *IEEE Signal Processing Magazine*, vol. 26, no. 6, pp. 124–135, Nov. 2009.

[11] J. Kim, J. Chong, and I. Lane, "Efficient On-The-Fly Hypothesis Rescoring in a Hybrid GPU/CPU-based Large Vocabulary Continuous Speech Recognition Engine," in *Proc. Interspeech*, Sep. 2012, pp. 1183–1186.

[12] K. Hacioglu and B. Pellom, "A Distributed Architecture for Robust Automatic Speech Recognition," in *Acoustics, Speech, and Signal Processing, 2003. Proceedings. (ICASSP '03). 2003 IEEE International Conference on*, April 2003, vol. 1, pp. I–328–I–331 vol.1.

[13] J. Kim and W. Sung, "Multi-User Real-Time Speech Recognition with a GPU," in *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*, March 2012, pp. 1617–1620.

[14] NVIDIA, *CUDA C Programming Guide version 7.5*, September 2015.

[15] A. Mohamed, G.E. Dahl, and G.E. Hinton, "Acoustic modeling using deep belief networks," *Audio, Speech, and Language Processing, IEEE Transactions on*, vol. 20, pp. 14–22, January 2012.

[16] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Processing Magazine*, pp. 82–97, 2012.