ACCELERATING 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@cs.cmu.edu

ABSTRACT

While prior works have demonstrated the effectiveness of Graphic-Processing Units (GPUs) for limited vocabulary speech recognition, these methods were unsuitable for recognition with large language models. To overcome this limitation, previously we introduced a novel *"on-the-fly rescoring"* approach in which search was performed over a WFST-network composed with a unigram language model on the GPU, and partial hypotheses were rescored on-the-fly using a large language model stored on the CPU. In this paper, we extend our previous algorithm to enable on-the-fly rescoring to be performed over an H-level network composed with any n-gram language model, and show that using a longer language model history in the H-level network improves decoding speed. We demonstrate that large language models can be applied on-the-fly with no degradation in decoding speed, realizing a LVCSR system that performs recognition over $22\times$ faster than a CPU implementation with no loss in recognition accuracy.

Index Terms— Large Vocabulary Continuous Speech Recognition (LVCSR), Weighted Finite State Transducer (WFST), Graphics Processing Units (GPU)

1. INTRODUCTION

In order to obtain high recognition accuracy, state-of-the-art speech recognition systems for tasks such as broadcast news transcription [1, 2] or voice search [3, 4] perform recognition with large vocabularies ($>$ 1 million words), large acoustic models (millions of model parameters), and extremely large language models (billions of n-gram entries). While these models can be applied in offline speech recognition tasks, they are impractical for real-time speech recognition due to the large computational cost required during decoding. While prior works [5, 6] have demonstrated the efficiency of using many-core graphics processing unit (GPU) in speech recognition for limited vocabulary tasks. These approaches, however, are not suitable for large vocabulary tasks as the limited memory on the GPU architecture becomes a significant bottleneck when large acoustic and language models are applied during recognition.

The most significant challenge is handling the extremely large language models used in modern large vocabulary speech recognition systems [1, 2, 4]. These models can contain millions of unique vocabulary entries, billions of n-gram contexts, and can easily require 20 GB or more to be stored in memory. Significantly pruned language models could fit within the limited memory available on GPU platforms ($<$ 6 GB), however, the speech recognition accuracy would be severely degraded.

In [7], we proposed a novel on-the-fly rescoring algorithm specifically optimized for GPU-accelerated platforms. In this approach a n-best Viterbi search was performed on the GPU using a WFST composed with a unigram language model. Partial hypotheses generated during search were rescored on-the-fly using a large language model stored on the CPU. As the search network was composed with a unigram language model, this earlier implementation was limited in two significant ways. First, as the rescoring weight was precomputed within the language model applied during rescoring it could only be applied to fully composed H-level networks to those composed with a unigram language model. Second, the decoding speed was limited due to the large number of language model look-ups required on the CPU when speech recognition was performed with large language models.

In this paper, we extend our previous work in two ways. First, we extend our on-the-fly rescoring algorithm to enable search to be performed on any H-level WFST network. Second, we extend our implementation to leverage multi-core CPUs reducing the time required for language model look-up. Using these approaches we are able to realize a GPU-accelerated speech recognition engine that can efficiently incorporate large language models on-the-fly with no degradation in decoding speed.

2. WFST-BASED SPEECH RECOGNITION ON GPU-ACCELERATED PLATFORM

A WFST is a type of finite state automaton that describes a transduction from one symbol sequence to another, with input symbols $i[\cdot]$ and output symbols $o[\cdot]$, as well as a weight $w[\cdot]$ defined on each of its arcs in WFST N. In the WFST framework, we treat the speech recognition problem as finding the minimally weighted transduction from input speech signal O to word sequence W. The overall weight $\Omega(O \rightarrow W)$ on a WFST N transduces O into W can be written as:

$$\Omega(O \rightarrow \hat{W}) = \min_{W} \left\{ \min_{f \in \Pi_N(O \rightarrow W)} w[f] \right\} \quad (1)$$

where $\hat{W}$ indicates the word sequence on the minimally weighed path, $\Pi_N(O \rightarrow W)$ denotes a set of complete paths such as f that accept O and output W in N as derived in [3].

One of the advantages of using the WFST framework for speech recognition is the unified framework used for representing different knowledge sources, such as Hidden Markov Models (HMMs) H, context-dependency phoneme model C, lexical or word pro-

The author would like to thank Jike Chong for his discussions and insight on the parallel *"merge-and-sort"* algorithm. This work is supported in part by the Samsung GRO program

nunciation model L, and n-gram language model G_n with weight $-\log P(O|V), 0, -\log P(V|W), -\log P(W)$ respectively, where V is a phoneme sequence. These knowledge sources can be composed together and optimized for speed and size ahead of decoding [8]. In this work, we use the fully composed HMM state level (H-level) WFST network

$$N_n = H \circ C \circ L \circ G_n \tag{2}$$

where "$\circ$" is a composition operator.

2.1. Standard decoding (one-pass strategy)

In a standard time-synchronous Viterbi search, an empty path or hypothesis f is assign to the initial state of N and generates a new hypothesis f' by adding a transition e originating from the final state of f, $n[f]$ for each time frame. The accumulated weight $w[f']$ can be calculated as

$$w[f'] = w[f] + w[e] \tag{3}$$

when we use the negative log likelihood as weights. Viterbi search selects the best hypothesis (incoming path with minimum weight) from competing hypotheses that meet at the same state in N. The search process over an H-level WFST consists of two phases. First, weights from observation likelihoods are computed for each active state within the search space (Phase 1). Next, traversal over the WFST is performed guided by computed weights and the WFST graph structure (Phase 2)

Prior works [9, 10, 11] implemented speech recognition on GPU-accelerated platforms. In these works, observation likelihoods in Phase 1 were computed on the GPU, and Phase 2, graph traversal was performed on the CPU. The speed-up obtained with this implementation was limited due to the overhead of communicating intermediate results between Phase 1 and 2 at every time step. [5, 6, 12] implemented both Phase 1 and 2 on the GPU to eliminate unnecessary data transfer between the CPU and the GPU. This enabled more than an order of magnitude speed-up compared to an optimized sequential implementation. However, the language models that could be applied during decoding were limited to WFSTs which can be fitted within the local GPU memory ($\sim$ 6GB).

2.2. Lattice rescoring (two-pass strategy)

One typical approach of applying an extremely large language model to speech recognition is generating a lattice first using a lower order pruned language model and then conducting lattice rescoring using a significantly larger model [13]. In the WFST framework, the lattice generated from the first path, lower and higher order language model are represented as WFST, T, G_l, G_h respectively. The rescored lattice can be composed as:

$$T' = det(T \circ (-G_l)) \bullet G_h \tag{4}$$

where det is *determinization* operation as described in [8] and "$\bullet$" is a special composition operation by matching a ϕ symbol that indicates back-off arcs in G [14]. Finding the minimally weighted path from T' is effectively finding $\hat{W}$ has weight as

$$\begin{aligned}\Omega(O \rightarrow \hat{W}) = \min_{W} \Big\{ & \min_{h \in \Pi_N(O \rightarrow W)} w_N[h] \\ & - \min_{f \in \Pi_{G_l}(W)} w_{G_l}[f] + \min_{g \in \Pi_{G_h}(W)} w_{G_h}[g] \Big\}.\end{aligned} \tag{5}$$

Note that $\Pi_G(W)[\cdot]$ denotes a set of complete paths accepting the word sequence W.

Generating the lattice is usually slower than the standard Viterbi decoding since the decoder must keep all sub-optimal forward links at each frame rather than only the best backward link required to find the best path. Additionally, on GPU-accelerated platforms, these sub-optimal forward links must be copied from the GPU to the CPU for each time frame. This communication overhead makes the lattice generation inefficient on GPU-accelerated platforms. In this paper, we compare the performance of the lattice rescoring to the proposed algorithm. We generate lattices following the approach described in [15]. WFST search is performed on the GPU, while the lattice is generated dynamically on the CPU.

2.3. On-the-fly composition (one-pass strategy)

On-the-fly composition approaches dynamically build a WFST on-the-fly during decoding with multiple WFSTs [16, 17, 18, 19]. The most common approach is to decouple the language model from the other components. A WFST such as, $C \circ det(L)$ (and possible the acoustic model H) is composed offline. Then in the recognition phase, search performs an additional on-demand composition of the $C \circ det(L)$ with G during decoding.

This method has been shown to be economical in terms of memory without any accuracy degradation. In this approach, however, all WFSTs should be stored on the GPU memory for the on-demand composition. In this paper, we do not compare the on-the-fly composition algorithm to the proposed algorithm as it does not solve the problem of limited GPU memory during search.

2.4. On-the-fly rescoring (one-pass strategy)

In this approach, partial hypotheses or lattices are generated from N_l, but they are weighted based on (5) during a time synchronous Viterbi search using a higher order language model to solve (5) in a one-pass strategy [3, 7, 20]. The main difference from two-pass strategy is the decoder uses all knowledge sources during search. This is effective for both correct path selection and pruning [3] and obtains faster decoding speeds compared with on-demand composition while using less memory when compared with standard decoding [21].

[3, 7, 20] use a similar concept for on-the-fly rescoring, however the purpose, implementation and target platform are different. [3] generates partial hypotheses during search N_l, and produce co-hypotheses by matching symbol sequences between N_l and $G_{l/h}$ similar to the on-the-fly composition approach only when the hypotheses outputs a word symbol. This approach requires a fully composed WFST N_h before factoring N_h into N_l and $G_{l/h}$ which may not be feasible. Additionally, the language model should be represented as WFST which is not possible for some language models (i.e. neural network language models).

Compared to [3], [20] provides a more general rescoring framework. This approach conducts n-best search over N_l generating a lattice which contains only the resulting acoustic model scores. This resulting lattice is then rescored using a higher order language model to generate a rescored lattice on-the-fly. This approach assumed that the overall weight $\Omega(O \rightarrow \hat{W})$ consisted only of weights from acoustic model and the language model. This may cause an accuracy degradation if the word pronunciation transducer L has different weight, $-\log P(V|W)$, across different pronunciations of a word.

In [7], we proposed an efficient on-the-fly rescoring algorithm specifically developed for GPUs which allowed extremely large language models to be applied while maintaining the superior throughput of a GPU-accelerated implementation. This method conducted n-best decoding similar to [20], but did not omit any language model

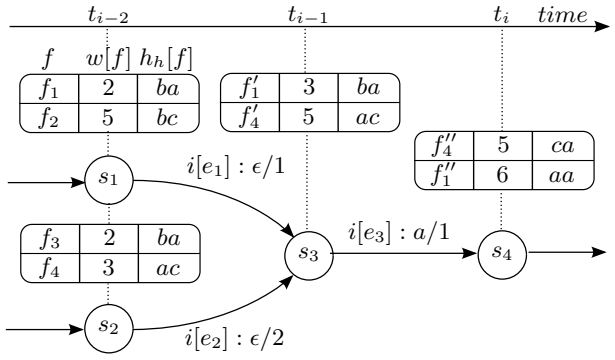

Fig. 1. Example of the proposed algorithm when N_2 is rescored with 3-gram language model ($w_R(f_1', a) = 2$, $w_R(f_4', a) = -1$).

weights. However, the approach was limited to a unigram language model on the GPU and the different computation between unigram and higher order language model weight on CPU limiting the overall decoder speed.

3. GENERALIZED ON-THE-FLY RESCORING

In our proposed method, n-best time-synchronous Viterbi search is performed on the GPU using N_l which is composed using lower order language model. Higher order language models are stored and accessed on the CPU during search. In this method, a new hypothesis f' is generated by adding a transition e originating from $n[f]$ for each time frame regardless of $o[e]$. If transition e outputs nothing ($o[e] = \epsilon$ and $o[f'] = o[f]$), rescoring is not applied and follows the standard decoding as explained in Section 2.1. Only when the transition e outputs a non-epilson symbol ($o[e] \neq \epsilon$ and $o[f'] \neq o[f]$), partial hypothesis f' is rescored. The weight of f' can be calculated as

$$w[f'] = w[f] + w[e] + w_R(f, o[e]) \tag{6}$$

where w_R is a rescoring weight. We can calculate the rescoring weight by subtracting the lower order language model weight from the higher order language model on the CPU corresponding to (5) as

$$w_R(f, o[e]) = -\log P(o[e]|h_l[f]) - \Big\{ -\log P(o[e]|h_h[f]) \Big\} \tag{7}$$

where $h_l[f]$, $h_h[f]$ is a recent word history of f for the lower and higher order language model, respectively.

In the proposed algorithm, n-best Viterbi search is conducted by assigning a n-best hypotheses list to each arc and state. When multiple arcs meet at the same state, n-best hypotheses lists are merged and the Viterbi search chooses n minimally weighted hypotheses across all n-best hypotheses lists. In Fig. 1, f_1' from state s_1 and f_4' from state s_2 are maintained and sorted based on their total weight. In addition, if hypotheses have same higher order history, only less weighted hypothesis will be maintained. For example, f_3' is pruned because this hypothesis has same higher order history "ba" but higher weight compared with f_1' when these are meet in s_3.

Keeping the n-best list is important to achieve comparable accuracy [7]. If we choose the best hypothesis as explained in Section 2.1, the partial hypothesis of the final best hypothesis $\hat{f}$ can be pruned out before it reach to an transition e with a non-epsilon output where a hypothesis can be rescored. This scenario is illustrated in Fig. 1 where the hypothesis f_1'' is the original best path using the standard decoding approach and f_4'' is the rescored best path. If we only kept the best hypothesis, the partial hypothesis f_4' of the rescored best path f_4'' would be pruned in s_3 and could not reach to the arc e_3 between s_3 and s_4 where the rescoring would be applied.

The maximum size of the n-best list is determined carefully based on the size of the vocabulary and the order of the language model. For a given a vocabulary size, a WFST composed with lower order language model requires a larger n compared to that composed with higher order language model to achieve comparable accuracy. Similarly, a WFST that has a larger vocabulary requires a larger n-best list to retain partial hypotheses until they reach a rescoring point.

Algorithm 1 Parallel n-best list merge and sort algorithm.

```
1:  while i, j < n do                                  ▷ For each GPU thread
2:      g'_j ← g_j
3:      if w[f_i] > w[g_j] then
4:          increment j
5:      else
6:          if h[f_i] ∉ h[g] then
7:              g''_j ← atomicCAS(g_j, g'_j, f_i)      ▷ Returns current g_j
8:              if g''_j = g'_j then
9:                  Insert(f, g_j)
10:                 increment i, j
11:             end if
12:         else
13:             increment i
14:         end if
15:     end if
16: end while
```

4. OPTIMIZATION DETAILS

4.1. Parallel n-best list *merge-and-sort* on the many-core GPUs

The most important parallelization challenge during the n-best Viterbi search on the many-core GPU is the merging of n-best list on reconvergent paths. There may be hundreds of arcs, each with an n-best list, trying to write into the n-best list at the destination state at a same time. We handled this challenge by developing a new technique to merge the n-best list atomically on a highly parallel platform using atomic Compare-And-Swap (atomicCAS) operations [22]. The GPU provides hardware supported atomic operations that are efficiently implemented. We leverage this capability to implement our n-best parallel *"merge-and-sort"* algorithm on this platform as described in Algorithm 1.

4.2. Parallel language model look-up on the multi-core CPU

In the proposed algorithm, the language model look-up phase is conducted on CPU which consists of following steps:

1. Copy $h[f]$, $o[e]$ from GPU to CPU.
2. Compute $w_R(f, o[e])$ following (7) on CPU.
3. Update $h[f']$, $w_R(f, o[e])$ from CPU to GPU.

In our previous work, we observed that with large vocabularies the execution time of language model look-up on the CPU increased rapidly compared to the other phase [7]. To resolve this problem, we parallelize step 2 above using OpenMP [23]. Furthermore, as we are using physically independent hardware, it is possible to completely hide the latency of the language model look-up by executing the language model lookup (on CPU), and acoustic score computation (on GPU) concurrently.

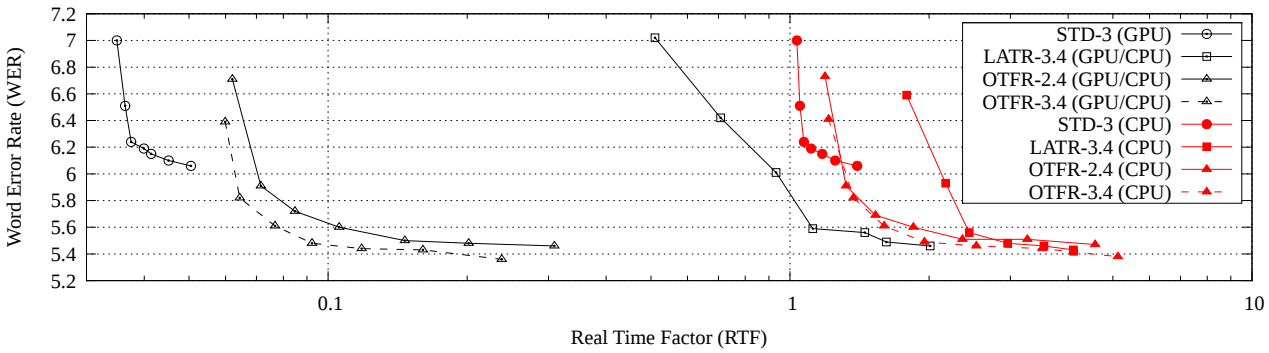

Fig. 2. Relationship between word error rate and Real Time Factor, ($n_{2.4} = 5$, $n_{3.4} = 3$, $m_{2.3} = m_{3.4} = 12$ CPU cores)

Vocab.	n-gram	# of n-gram	WFST (MB)	LM binary (MB)
1M	1	1.0M	91	1
	2 (Pruned)	15.1M	2,342	357
	3 (Pruned)	10.1M	3,583	407
	4	769.9M	-	19,554

Table 1. Size of WFSTs and binary language models.

Phase	Baseline	+OpenMP	Proposed
Active Set Preparation	0.03	0.03	0.03
Observation likelihood computation	0.02	0.02	0.02
Language Model Look-up	0.39	0.04	0.03
WFST Search	0.01	0.01	0.01

Table 2. Processing time (RTF) per phase. ($n_{3.4} = 3$, $m_{3.4} = 12$ CPU cores)

5. EXPERIMENTAL EVALUATION

We evaluated the effectiveness of our proposed "*generalized on-the-fly rescoring*" algorithm on a large vocabulary version of the WSJ task. We used a combined evaluation set consisting of the November 1992 ARPA WSJ 5k evaluation set (330 sentences) and the November 1993 ARPA WSJ 20k evaluation set (213 sentences, open vocabulary). A deep neural network (DNN) acoustic model was trained using the WSJ data set with 39 dimensional MFCCs feats with a LDA transformation as described in [24, 25]. The resulting acoustic model contained 2,946 phonetic states and consisted of 4 hidden layers, each with 2,000 hidden units.

WFSTs were composed and optimized offline for efficient parallel time-synchronous graph traversal on GPU-accelerated platform as described in [5, 6]. Table 1 shows the size of the final fully-composed WFSTs for pruned 1 Million vocabulary language models. A vocabulary of 1 Million words and a 4-gram language model with 770 Million entries was applied during decoding. We evaluate the proposed algorithm using a single GPU in a 8-way NVIDIA Tesla C2075M GPU server with two Intel Xeon E5-2640 6-core CPUs. The NVIDIA Tesla C2075M GPU contains 448 CUDA cores and 6GB GDDR5 memory.

We compared three decoding schemes: Standard decoding using a WFST composed with a 3-gram language model (STD-3), lattice rescoring where the resulting 3-gram lattice was rescored with a 4-gram language model(LATR-3.4), and the proposed on-the-fly rescoring approach where 3-gram partial hypotheses were rescored during search with a 4-gram language model (OTFR-3.4).

5.1. Recognition accuracy

First, we validated the accuracy of the proposed approach (OTFR) by comparing it's performance to standard lattice rescoring (LATR) techniques. We generated 2 fully composed WFSTs using the same knowledge sources but different, highly pruned, $\{2, 3\}$-gram language models (N_2 and N_3) . Decoding with N_2, N_3 and rescoring on-the-fly using a 4-gram language model obtained the same 5.5% Word Error Rate (WER) as LATR-3.4. During decoding n-best lists of size 5 and 3 were used for the bigram and trigram respectively. For the bigram case a larger n is required to to achieve comparable WER, as explained in Section 3.

5.2. Decoding Speed

Next, we evaluated decoding speed using both the single core CPU implementation and our proposed heterogeneous CPU-GPU implementation. The baseline CPU implementation is optimized using Automatically Tuned Linear Algebra Software (ATLAS) to accelerate the acoustic weight computation process [26].

In Table 2, the baseline shows decoding speed of the proposed algorithm without the optimization explained in Section 4.2. The language model look-up consumed the majority of the decoding time. By performing language model look-up using 12 CPU cores the time required for look-up is reduced by factor of 9.8× from 0.39 to 0.04 RTF. Finally, by performing language model look-up concurrently with the acoustic weight computation on the GPU the decoding time is reduced further from 0.11 RTF to 0.09 RTF.

In the OTRF-3.4 case, decoding was performed, 11× faster than real-time when the WER was 5.5%. This was 22× faster than a highly optimized single CPU implementation. OTFR-3.4 was 40% faster (0.11 vs. 0.24 RTF) than OTFR-2.4 indicating that a WFST composed with the higher order language model converges faster than the lower order case.

6. CONCLUSION

In this paper we proposed a generalized on-the-fly rescoring approach to enable efficient decoding with extremely large language models on the heterogeneous CPU-GPU platforms. The proposed approach enables search to be performed on an H-level WFST network composed with an n-gram language models of any length. By parallelizing language model look-up on the CPU and implementing a parallel n-best list merge-and-sort on the GPU, our LVCSR system was able to realize recognition with a 1 Million word vocabulary at 11× faster than real-time, approximately 22× faster than a single-threaded CPU implementation.

7. REFERENCES

[1] R. Hsiao, M. Fuhs, Q. Jin Y. Tam, Ian Lane, and T. Schultz, *CMU/InterACT Mandarin Speech Recognition System for GALE. Handbook of Natural Language Processing and Machine Translation*, chapter 3.5.3, pp. 496–504, Springer, 2011, ISBN 978-1-4419-7712-0.

[2] U. Nallasamy, Ian Lane, M. Fuhs, M. Noamany, Y. Tam, Q. Jin, and T. Schultz, *CMU/InterACT Arabic Speech Recognition System for GALE. Handbook of Natural Language Processing and Machine Translation*, chapter 3.6.4, pp. 535–540, Springer, 2011, ISBN 978-1-4419-7712-0.

[3] T. Hori, C. Hori, Y. Minami, and A. Nakamura, "Efficient WFST-Based One-Pass Decoding With On-The-Fly Hypothesis Rescoring in Extremely Large Vocabulary Continuous Speech Recognition," *Audio, Speech, and Language Processing, IEEE Transactions on*, vol. 15, no. 4, pp. 1352 –1365, may 2007.

[4] J. Schalkwyk, D. Beeferman, F. Beaufays, B. Byrne, C. Chelba, M. Cohen, M. Kamvar, and B. Stropek, *Google Search by Voice: A case study*, Springer, 2010.

[5] 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.

[6] 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.

[7] 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.

[8] M. Mohri, F. Pereira, and M. Riley, "Weighted Finite-State Transducers in Speech Recognition," *Computer Speech and Language*, vol. 16, no. 1, pp. 69–88, 2002.

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

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

[11] 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.

[12] 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.

[13] A. Ljolje, F. Pereira, and M. Riley, "Efficient general lattice generation and rescoring," in *Proc. Eurospeech*, 1999.

[14] M. Riley, C. Allauzen, and M. Jansche, "Openfst: An open-source, weighted finite-state transducer library and its applications to speech and language," in *HLT-NAACL (Tutorial Abstracts)*, 2009, pp. 9–10.

[15] D. Povey, M. Hannemann, G. Boulianne, L. Burget, A. Ghoshal, M. Janda, M. Karafiát, S. Kombrink, P. Motlícek, Y. Qian, K. Riedhammer, K. Veselý, and N. T. Vu, "Generating exact lattices in the wfst framework," in *Proc. ICASSP*, 2012, pp. 4213–4216.

[16] D. Caseiro and I. Trancoso, "Using dynamic wfst composition for recognizing broadcast news," in *Proc. Interspeech*, 2002.

[17] D. Caseiro and I. Trancoso, "A specialized on-the-fly algorithm for lexicon and language model composition," *IEEE Trans. on Audio, Speech & Language Processing*, vol. 14, no. 4, pp. 1281–1291, 2006.

[18] T. Hori and A. Nakamura, "Generalized fast on-the-fly composition algorithm for wfst-based speech recognition," in *Proc. Interspeech*, 2005, pp. 557–560.

[19] T. Hori, C. Hori, and Y. Minami, "Fast on-the-fly composition for weighted finite-state transducers in 1.8 million-word vocabulary continuous speech recognition," in *Proc. Interspeech*, 2004.

[20] H. Sak, M. Saraclar, and T. Güngör, "On-the-fly lattice rescoring for real-time automatic speech recognition," in *Proc. Interspeech*, 2010, pp. 2450–2453.

[21] P. R. Dixon, C. Hori, and H. Kashioka, "A comparison of dynamic wfst decoding approaches," in *Proc. ICASSP*, 2012, pp. 4209–4212.

[22] NVIDIA, *NVIDIA CUDA Programming Guide Version 5.0*, May 2012.

[23] L. Dagum and R. Menon, "Openmp: an industry standard api for shared-memory programming," *Computational Science & Engineering, IEEE*, vol. 5, no. 1, pp. 46–55, 1998.

[24] 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.

[25] 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.

[26] C. W. Antoine, A. Petitet, and J. J. Dongarra, "Automated empirical optimization of software and the atlas project," *Parallel Computing*, vol. 27, pp. 2000, 2001.