1. Motivation and General Ideas

Recent advances in deep learning based large vocabulary continuous speech recognition (LVCSR) invoke growing demands in large scale speech transcription. The inference process of a speech recognizer is to find a sequence of labels whose corresponding acoustic and language models best match the input feature [1]. The main computation includes two stages: acoustic model (AM) inference and linguistic search (weighted finite-state transducer, WFST). Large computational overheads of both stages hamper the wide application of LVCSR.

To reduce computation of the first stage, researchers have proposed a variety of efficient forms of AMs, including novel structures [2, 3], quantization [4, 5] and frame-skipping [6]. Meanwhile, algorithmic improvement of linguistic search, e.g. pruning [7], rescoring [8] and lookahead [9, 10], was the mainstream approach to speed up the second stage in the past.

This work mainly focuses on linguistic search. Despite the WFST based LVCSR approach has been improved for several decades, two fundamental deficiencies remain: i) The WFST search space is large and its graph traversal algorithms are conducted at each frame, e.g. frame synchronous Viterbi beam search space is large and its graph traversal algorithms are conducted at each frame, e.g. frame synchronous Viterbi beam search (FSD). ii) Most of these algorithms are originally serial algorithms and parallelizing them is non-trivial.

Benefit from stronger classifiers, deep learning, and more powerful computing devices, we propose general ideas and some initial trials to solve these fundamental problems:

- **Reduce the search complexity by end-to-end modeling**
  Recent advances in more potent neural networks enable stronger modeling effects in the context and the history of the sequential modeling [11, 12, 13, 14, 15]. More labeled data further alleviates the sparseness and generalization problem in the modeling. Thus, it is promising to decompose the sequence into larger model granularities. Research has been conducted on different model granularities from frame level to the whole sequence [16, 17, 18, 19, 20]. e.g., in [17], a word level deep learning based acoustic model is trained on 125K hours labeled data and outperforms models with smaller granularity. We propose to change the frequency of Viterbi search from feature frame to each label output. Correspondingly, a post-processing is applied on the frame level acoustic model outputs. Based on this framework, the larger model granularities we take, the less search complexity we obtain.

- **Accelerate the search speed using parallel computing**
  GPU-based parallel computing is another potential direction which utilizes a large number of units to parallelize the computation. As common language models (LM) can be expressed as WFSTs, the idea is to parallelize WFST graph traversal algorithms. Our initial work parallelizes Viterbi algorithm [21] and redesigns it to fully utilize parallel computing devices nowadays, e.g. Graphics Processing Units (GPU) and Field-Programmable Gate Arrays (FPGA). To utilize large LMs in the 2nd pass and support rich post-processing, our design is to decode the WFSTs and generate exact lattices [22]. The decoder remains to be general-purpose and does not impose special requirements on the form of AM or LM. The ideas to apply GPU parallel computing in WFST decoding include: i) Abstract the dynamic programming in the Viterbi algorithm as thread synchronization using atomic GPU operations. ii) Propose a load balancing strategy for parallel WFST search scheduling among GPU threads. iii) Parallelize exact lattice generation and pruning algorithms. Similar ideas can be applied to other WFST algorithms, e.g. determination and minimization [23], and speedups of them can be expected.

2. Label Synchronous Framework

The search process is proposed to change from the feature level to the label level, i.e. label synchronous decoding (LSD) [24]. Within the label inference, a post-processing is applied on the frame level acoustic model outputs. The formulation and implementation are discussed on CTC [25] and LF-MMI [26].

During inference stage, Viterbi beam search of CTC model [25] can be expressed as,

\[
\mathbf{w}^* = \arg\max_w \left\{ P(w) \prod_{l \in \mathcal{U}} \frac{P(l|\mathbf{x})}{P(l|w)} \right\} \tag{1}
\]

where \(\mathbf{x}\) is the feature sequence, \(w\) is a word sequence and \(\mathbf{w}^*\) is the best word sequence. \(P(l|\mathbf{x})\) denotes the label sequence, e.g. the phoneme sequence, corresponding to \(w\). Within the calculation of \(P(l|\mathbf{x})\), a post-processing is proposed on the frame level neural network outputs, \(P(\pi|x)\). Here, the set of common blank frames are defined as: \(\mathcal{U} = \{ u : y^u_{\text{blank}} > \theta \}\), where \(y^u_{\text{blank}}\) is the probability of the blank unit at frame \(u\). With a softmax layer in the CTC model, if the blank acoustic score is large enough and approaching a constant of 1, it can be regarded that all competing paths share the same span of the blank frame. Thus ignoring the scores of the frame does not affect the acoustic score rank in decoding:

\[
P(l|\mathbf{x}) \equiv \sum_{\pi \in \mathcal{B}(\mathbf{l})} P(\pi|\mathbf{x}) \approx \sum_{\pi \in \mathcal{B}(\mathbf{l})} \prod_{u \in \mathcal{U}} y^u_{\pi_u} \tag{2}
\]

where \(\mathcal{B}\) is a one-to-many mapping between labels, e.g. phonemes, and CTC states. LSD is summarized as Algorithm[1]. The main difference compared with FSD Viterbi algorithm is the introduction of isBlankFrame(\(F\)) to detect whether a frame is blank or not. Recently, novel HMM topology was proposed in [27, 28], which holds a similar one-to-many mapping as \(\mathcal{B}\) function of CTC. The three-state HMM contains a state...
simulating the function of blank. Thus similar post-processing as Equation 4 and its corresponding algorithm can be derived.

Algorithm 1 Label Synchronous Viterbi Beam Search for CTC
(Inputs: start and end nodes, token queue, time frames)

1: procedure LSD FOR CTC (S, E, Q, T)
2: \( Q \leftarrow S \) \( \triangleright \) initialization with start node
3: for each \( t \in [1, T] \) do \( \triangleright \) frame-wise NN Propagation
4: \( F \leftarrow N N_{\text{Propagate}}(t) \)
5: if \( \text{IsBlankFrame}(F) \) then \( \triangleright \) phone-wise decoding
6: \( Q \leftarrow \text{ViterbiBeamSearch}(F, Q) \)
7: \( \hat{B} \leftarrow \text{finalTransition}(E, S, Q) \) \( \triangleright \) to reach end node
8: \( \text{backtrace}(\hat{B}) \)

The decoding complexity reduction from FSD to LSD is as Equation 5, where \( T \) is the number of frames, \(|L'| |W|\) are sizes of label set and vocabulary. The number of blank frames, \(|U|\), is always approaching \( T \). Thus FSD is greatly sped up.

\[
C \propto T \cdot |L'| \cdot |W| \Rightarrow (T - |U|) \cdot |L'| \cdot |W| \quad (3)
\]

Experiments on Switchboard [27] show the speedup as Figure 1.

![Figure 1: hub5e-swbb WER versus Search Real-time Factor (RTF) in CTC Obtained by LSD and Other Pruning Methods.](image1)

3. GPU-based Parallel WFST Decoding

Figure 2 shows the framework of parallel Viterbi beam search [28]. The procedure of decoding is similar to the CPU version [29] but works in parallel with specific designs. Load balancing controls the thread parallelism over both WFST states and arcs. Two GPU concurrent streams perform decoding and lattice-pruning in parallel launched by CPU asynchronous calls.

We parallelize token passing algorithm [30] in two levels. Tokens in different states are processed parallelly. For each token, we traverse its out-going arcs in parallel as well. Because WFST states might have different numbers of out-going arcs, the allocation of states and arcs to threads can result in load imbalance. We use a dispatcher in charge of global scheduling, and make \( N \) threads as a group (\( N = 32 \)) to process arcs from a token. When the token is processed, the group requests a new token from the dispatcher. We implement task dispatching as an atomic operation [31]. Figure 2 shows an example.

At each frame, the Viterbi path is obtained by a token recombination procedure, where a min operation is performed on each state over all of its incoming arcs (e.g. state 7 in Figure 2 and the incoming arcs from state 2, 5 and 7), to compute the best cost and the corresponding predecessor of that state. We abstract this process as thread synchronization using atomic GPU operations. After finishing all synchronization, we aggregate survived tokens exploiting thread parallelism. We also parallelize exact lattice generation and pruning algorithms with similar ideas, described in [29].

![Figure 2: Example of Dynamic Load Balancing. The dashed box denotes a CUDA cooperative group and different groups are with different colors. Each group is controlled by Thread 0 of it. After processing all the forward links of a state, Thread 0 accesses the dispatcher and the next token is dynamically decided by atomic operation. Group 0 and 1 work in parallel.](image2)

![Figure 3: LM Size, Frame Rates and Architectures Comparison.](image3)

4. Future works

Our general ideas to reduce the computation of speech applications are two folds: reduce the search complexity by end-to-end modeling and accelerate WFST algorithms using parallel computing. For the first part, LSD in both discriminative and generative sequence models should be fully investigated. Integrating LSD as a sub-sampling module in the sequence-to-sequence framework is another direction [32]. Keyword spotting and confidence measures in this framework needs to be considered [33] [34] [35]. For the second part, the initial work in Viterbi decoding can be extended to most of WFST graph traversal algorithms [17]. According to specific application scenarios, parallel implementations of various devices (e.g. CPU, GPU and FPGA) can be considered. Neural network language models should be taken into account [36].

5. Acknowledgements

I express sincere gratitude to my advisors, Kai Yu and Yanmin Qian. I thank the speech technology group of AISpeech Ltd. for infrastructure support and valuable technical discussions. I thank my internship mentors, Jasha Droppo, Jinyu Li, Daniel Povey, Justin Luitjens, Christian Fuegen and Yongqiang Wang. I thank my internship mentors, Jasha Droppo, Jinyu Li, Daniel Povey, Justin Luitjens, Christian Fuegen and Yongqiang Wang. I thank the speech technology group of AISpeech Ltd. for infrastructure support and valuable technical discussions.
6. References


