Opportunities and Challenges of Parallelizing Speech Recognition

Jike Chong
UC Berkeley
Dept. of EECS
Berkeley, CA 94720, USA

Gerald Friedland, Adam Janin, Nelson Morgan, Chris Oei
International Computer Science Institute
1947 Center Street, Suite 600
Berkeley, CA 94530

Abstract

Automatic speech recognition enables a wide range of current and emerging applications such as automatic transcription, multimedia content analysis, and natural human-computer interfaces. This article provides a glimpse of the opportunities and challenges that parallelism provides for automatic speech recognition and related application research from the point of view of speech researchers. The increasing parallelism in computing platforms opens three major possibilities for speech recognition systems: improving recognition accuracy in non-ideal, everyday noisy environments; increasing recognition throughput in batch processing of speech data; and reducing recognition latency in real-time usage scenarios. We describe technical challenges, approaches we've taken, and possible directions for future research to guide the design of efficient parallel software and hardware infrastructures.

1 Introduction

We have entered an era where applications can no longer rely on significant increases in processor clock rate for performance improvements, as clock rate is now limited by factors such as power dissipation [4]. Rather, parallel scalability (the ability for an application to efficiently utilize an increasing number of processing elements) is now required for software to obtain sustained performance improvements on successive generations of processors.

Automatic Speech Recognition (ASR) is an application that consistently exploits advances in computation capabilities. With the availability of a new generation of highly parallel single-chip computation platforms, ASR researchers are faced with the question: If you had unlimited computing, how could you leverage it to make speech recognition better? The goal of the work reported here is to explore plausible approaches to improve ASR in three ways:

1. Improve accuracy: Account for noisy and reverberant environments in which current systems perform poorly, thereby increasing the range of scenarios where speech technology can be an effective solution.
2. Improve throughput: Allow batch processing of the speech recognition task to execute as efficiently as possible, thereby increasing the utility for call centers and multimedia search and retrieval.
3. Improve latency: Allow speech-based applications, such as speech-to-speech translation, to achieve real-time performance, where speech recognition is just one component of the application.

This article discusses our current work as well as opportunities and challenges in these areas with regard to parallelization from the point of view of speech researchers.

2 Improving Accuracy

Speech recognition systems can be sufficiently accurate when trained with enough data having similar characteristics to the test conditions. However, there still remain many circumstances in which recognition accuracy is quite poor. These include moderately to seriously noisy or reverberant noise conditions, and any variability between training and recognition conditions with respect to channel and speaker characteristics (such as style, emotion, topic, accent, and language).

One approach that is both "embarrassingly" parallel and effective in improving ASR robustness is the so-called multistream approach. As has been shown for a number of years [5, 6, 15, 11], incorporating multiple feature sets consistently improves performance for both small and large ASR tasks. And as noted in [23], recent results have demonstrated that a larger number of feature representations can be particularly effective in the case of noisy speech. In order to conduct research on a massively parallel front end, a large feature space is desired. One approach that we and others have found to be useful is to compute spectro-temporal features. These features correspond to the output of filters that are tuned to certain rates of change in the time and frequency dimensions, and are inspired by studies in neuroscience, which have revealed that neurons in the mammalian auditory cortex are highly tuned to specific spectro-temporal modulations [9, 16]. Various approaches have been devised to combine and select the inherently large number of potential spectro-temporal features because processing them entirely is currently considered computationally intractable.

2.1 Current Approach

Our current preferred approach to robust feature extraction is to generate many feature streams with different spectro-temporal properties. For instance, some streams might be more sensitive to speech that varies at a slow syllabic rate (e.g., 2 per second) and others might be more sensitive to signals that vary at a higher rate (such as 6 syllables per second). The streams are processed by neural networks (Multi-Layer Perceptrons, or MLPs) trained for discrimination between phones and generate estimates of posterior phone probability distributions. A critical theoretical and experimental question is how a large number of such streams should be best combined. For MLP-based feature streams, the most common combining techniques are: (1) appending all features to a single stream; (2) combining posterior distributions by a product rule, with or without scaling; (3) combining posterior distributions by an additive rule, with or without scaling; and (4) combining posterior distributions by another MLP, which may also use other features. When scaling is used for (2) or (3), there are open questions on how to do the scaling.

Our current best approach to combination is to train an additional Neural Network to generate combination weights by incorporating entropies from the streams as well as overall spectral information. We used a 28-stream system, including 16 streams from division of temporal modulation frequencies, 8 streams from division by spectral modulation frequencies, and 4 streams from a division by both [23]. Using this method, for the Numbers 95 corpus with the Aurora noises added [12], the average word error rate was 8.1%, reduced from 15.3% for MFCCs[1] and first and second order time derivatives. We have also run other pilot experiments that are encouraging. While robustness to environmental acoustics is our main focus, it was important to perform pilot experiments with both small and large vocabulary tasks using "clean" data, so that we could confirm that the particular form of expanded front end that we favored did not hurt us for tasks of different scales. Preliminary results have been obtained using a four-stream system on the Mandarin Broadcast news corpus used in DARPA GALE evaluations. In this case we used four equally weighted streams, with quasi-tonotopically divided spectro-temporal features. The system yielded a 13.3% relative improvement on the baseline, lowering word error rate from 25.5% to 22.1% The relative improvement in performance is lower than the 47% obtained for the Numbers95 corpus but it is comparable to what we have seen in other examples of moving techniques from small to large vocabulary tasks, particularly for similar cases where the training and test conditions are well matched.

2.2 Future Directions

In the current approach we apply the same modulation filters to the entire spectrum. Within this one feature stream, a pipe-and-filter parallel pattern can be used to distribute work across processing elements. Since the MLPs used within the stream depend on dense linear algebra, the wealth of methods to parallelize matrix operations can be exploited. We can also potentially expand the 28 streams to hundreds or thousands of streams by applying the Gabor filters to different parts of the spectrum as separate streams using a map-reduce parallel pattern.

We expect these techniques will be even more important to analyze speech from distant microphones at meetings, a task that naturally provides challenges due to noise and reverberation. Finally, there will be more parallelization considerations in combining the manystream methods with conventional approaches to noise robustness. As manystream feature combination naturally adapt to parallel computing architectures, we believe the improvement will be significant.

3 Improving Throughput

Batch speech transcription can be "embarrassingly parallel" by distributing different speech utterances to different machines. However, there is significant value in improving compute efficiency, which is increasingly relevant in today's energy limited and form-factor limited devices and compute facilities.

The many components of an ASR system can be partitioned into a feature extractor and an inference engine. The speech feature extractor collects feature vectors from input audio waveforms using a sequence of signal processing steps in a data flow framework. Many levels of parallelism can be exploited within a step, as well as across steps, as described in section 2.1. Thus feature extraction is highly scalable with respect to the parallel platform advances. However, parallelizing the inference engine requires surmounting significant challenges.

Our inference engine traverses a graph-based recognition network based on the Viterbi search algorithm [17] and infers the most likely word sequence based on the extracted speech features and the recognition network. In a typical recognition process, there are significant parallelization challenges in concurrently evaluating thousands of alternative interpretations of a speech utterance to find the most likely interpretation. The traversal is conducted over an irregular graph-based knowledge network and is controlled by a sequence of audio features known only at run time. Furthermore, the data working set changes dynamically during the traversal process and the algorithm requires frequent communication between concurrent tasks. These problem characteristics lead to

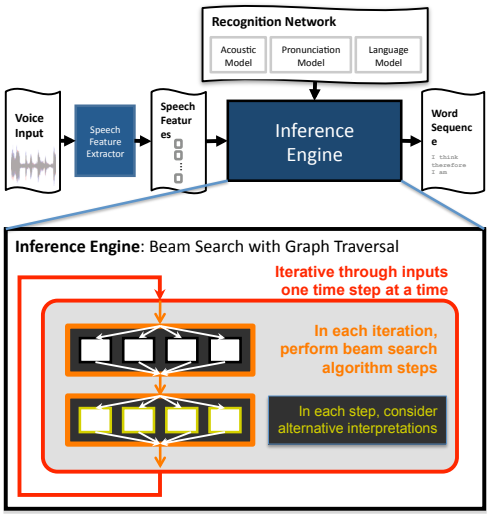

Figure 1: Decoder Architecture as described in Section 3.1.

unpredictable memory accesses and poor data locality and cause significant challenges in load balancing and efficient synchronization between processor cores.

There have been many attempts to parallelize speech recognition on emerging platforms, leveraging both fine-grained and coarse-grained concurrency in the application. Fine-grained concurrency was mapped onto the PLUS multiprocessor with distributed memory in [20]. The implementation statically mapped a carefully partitioned recognition network onto the multiprocessors to minimize load imbalance. [14] explored coarse-grained concurrency in speech recognition and implemented a pipeline of tasks on a cellphone-oriented multicore architecture. [22] proposed a parallel speech recognizer implementation on a commodity multicore system using OpenMP. The Viterbi search was parallelized by statically partitioning a tree-lexical search network across cores. The parallel recognition system proposed in [19] also uses a weighted finite state transducer (WFST) and data parallelism when traversing the recognition network. Prior works such as [10, 7] leveraged manycore processors and focused on speeding up the compute-intensive phase (i.e., observation probability computation) of ASR on manycore accelerators. Both [10, 7] demonstrated approximately 5x speedups in the compute-intensive phase and mapped the communication intensive phases (i.e., Viterbi search) onto the host processor. This software architecture incurs significant penalty for copying intermediate results between the host and the accelerator subsystem and does not expose the maximum potential of the performance capabilities of the platform.

3.1 Current Approach

More recently, we implemented a data-parallel automatic speech recognition inference engine on the NVIDIA GTX280 graphics processing unit (GPU), achieving over 11x speedup compared to SIMD optimized sequential implementation on an Intel core i7 CPU. With less than 8% sequential overhead, the solution promises more speedup on future more parallel platforms [8]. The speedup was enabled by constructing the recognition engine's software architecture to efficiently execute on single-chip manycore processors. There are four key implementation decisions that contributed to the speedup:

1. Exposing fine-grained parallelism: The software architecture of the inference engine is illustrated in Figure 1. The Hidden Markov model (HMM) based inference algorithm dictates that there is an outer iteration processing one input feature vector at a time. Within each iteration, there is a sequence of algorithmic steps implementing maximal-likelihood inference process. The parallelism of the application is inside each algorithmic steps, where the inference engine keeps track of thousands to tens of thousands of alternative interpretations of the input waveform. The challenge is that each algorithmic step only performs tens to hundreds of instructions on each alternative interpretation, thus synchronizations between the algorithmic steps impose sequential overheads. In multi-chip parallel platforms, the synchronization overhead significantly degrades parallel speedup. The opportunity brought by single-chip manycore parallel processors is that the synchronization overhead is significantly reduced to the point that the fine-grained parallelism can be exposed and the application speedup potentials can be realized.

2. Implementing all parts of an algorithm on the GPU: Current GPUs are accelerator subsystems managed by a CPU over the PCIe data bus. With close to a TeraFLOP of compute capability on the GPUs, moving operands and results between CPU and GPU can quickly become a performance bottleneck. As shown in Figure 1. In the inference engine, there is a compute intensive phase and a communication intensive phase of execution in each inference iteration. The compute intensive phase calculates the sum of differences of a feature vector against Gaussian mixtures in the acoustic model and can be readily parallelized. The communication intensive phase keeps track of thousands of alternative interpretations and manages their traversal through a complex finite state transducer representing the pronunciation and language models. While we achieved 17.7x speedup for the compute-intensive phase compared to sequential execution on the CPU, the communication-intensive phase is much more difficult to parallelize and received a 4.4x speedup. However, because the algorithm is completely

implemented on the GPU, we are not bottlenecked by the communication of intermediate results between phases over the PCI-express data bus, and have achieved a 11.3x speedup of the overall inference engine.

3. Leveraging fast hardware atomic operation support: The inference process is composed of data-parallel graph traversals on the recognition network. The graph traversal routines are executing in parallel on difference cores and frequently have to update the same memory location. This causes race conditions as the same piece of data must be read and conditionally written by multiple instruction streams at the same time. The race condition can be resolved using a sequence of data-parallel algorithmic steps in the application software or by using hardware-based atomic operation support. When leveraging hardware-based atomic operation support, however, the operations must be carefully managed as atomic operations to the same memory address are sequentialized. We leverage hardware atomic operation support at two levels: the core-level and the chip-level to avoid significant sequentialization of atomic operations.

4. Construct runtime data buffers to maximally regularize data access patterns: The recognition network is an irregular network and the traversal through the network is guided by user input available only at runtime. In each iteration of the inference engine, to maximally utilize the memory load and store bandwidth, we gather the data to be accessed during the iteration into a consecutive vector acting as runtime data buffers, such that the algorithmic steps in the iteration are able to load and store results one cache line at a time. This maximizes the utilization of the available data bandwidth to memory.

With these four key implementation decisions, we are able to overcome the parallelization challenges imposed by the application, and architect and implement a scalable parallel solution for speech recognition inference decoding.

3.2 Future Directions

The current work established an efficient software architecture for speech recognition targeting the highly parallel manycore platforms. Our ongoing work is constructing an application framework that allows many additional features to be extended without jeopardizing the efficiency and throughput of the implementation. One example of such additional feature can be an alternative observation likelihood computation that reduces the amount of computation necessary. Other improvements to the software architecture include producing word lattices or confusion-networks in the context of multiple-pass recognition systems. More generally, the throughput of a recognition engine can be further increased by distributing the workload to multiple processing nodes in a cluster of machines, where each machine can host multiple multicore and manycore processing units. The improvements in recognition throughput could also be used to trade off speed with accuracy, making viable approaches such as fast combination of results from multiple recognition engines with Recognizer Output Voting Error Reduction (ROVER) techniques.

4 Improving Latency

The parallelization of feature extraction and inference engine is being done as part of a larger goal of working with full applications in the Berkeley Parallel Computing Laboratory [18]. The focus is on providing useful implementations of driving applications that have real world latency requirements. Our application is called the "meeting diarist", in which users can access information from speech uttered in multiparty meetings during or shortly after the meeting. For speech recognition to be useful in multispeaker scenarios, it is also important to determine "who is speaking when", a process called "speaker diarization", and to further segment the speech in a way that is reasonable for human consumption. Ultimately we will be implementing and examining the entire application to better understand the sequential roadblocks to exploiting parallelism. This ongoing research is by no means complete but speaker diarization is a good example for explaining the opportunities to improve latency.

Most speaker diarization systems use agglomerative hierarchical clustering as a core approach to perform diarization. At a high-level, systems extract MFCC features from a given audio track, discriminate between speech and nonspeech regions (speech activity detection), and use the agglomerative clustering approach to perform both segmentation of the audio track into speaker-homogeneous time segments and the grouping of these segments into speaker-homogeneous clusters in one step. Speech activity regions are determined using a speech/non-speech detector, e.g., [21]. The non-speech regions are then excluded from the agglomerative clustering where the clustering is initialized using k clusters, with k larger than the number of speakers that are assumed to appear in the recording. Every cluster is modeled with a Gaussian Mixture Model containing g Gaussians. In order to train initial GMMs for the k speaker clusters an initial segmentation is generated by uniformly partitioning the audio into k segments of the same length. The ICSI system [1, 3] then performs the following iterations:

Re-Segmentation: Run Viterbi alignment to find the optimal path of frames and models. In the ICSI system, a minimum duration of 2.5 seconds is assumed for each speech segment. **Re-Training:** Given the new segmentation of the audio track, compute new Gaussian Mixture Models for each of the clusters. **Cluster Merging:**

Given the new GMMs, try to find the two clusters that most likely represent the same speaker. This is done by computing a score based on the Bayesian Information Criterion (BIC) of each of the clusters and the BIC score of a new GMM trained on the merged segments for two clusters. If the BIC score of the merged GMM is larger than or equal to the sum of the individual BIC scores, the two models are merged and the algorithm continues at the re-segmentation step using the merged GMM. If no pair is found, the algorithm stops.

As a result of different sequential optimization approaches [13], our current implementation runs at about $0.6\times$ realtime, i.e., for 10 minutes of audio data, diarization finishes in roughly 6 minutes. The main problem with the approach is that it requires the complete recording of a meeting file and so the latency is the time of the meeting + $0.6\times$ realtime of the meeting duration. There are many applications where online diarization is desirable and batch processing impractical.

4.1 Current Approach

An initial approach to online diarization was presented in the NIST Rich Transcription 2009 evaluations. The system consisted of a training step and an online recognition step. For the training step, we took the first 1000 seconds of the input and performed offline speaker diarization using the system described above. We then trained speaker models and a speech/non-speech model from the the output of the system. This is done by concatenating a random 60 second chunk of each speaker's segmented data and another one for the non-speech segments.

In the online recognition step, we recognize the remainder of the meeting using the trained models. The sampled audio data is noise-reduced and converted into MFCC features. For every frame, the likelihood for each set of features is computed against each set of Gaussian Mixtures obtained in the training step, i.e. each speaker model and the non-speech model. A total of 250 ten ms-frames is used for a majority vote on the likelihood values to determine the classification result. Therefore the latency totals at $t + 2.5\,s$ per decision (plus the portion of the offline training).

Such a system can significantly benefit from parallelism. If the offline diarization were two orders-of-magnitude faster than realtime, the offline diarization could process one minute of meeting in less than a second. The proposed online system could then run the offline system in the background constantly to update the models with the best solution found, taking into account the entire meeting so far.

4.2 Future Directions

Parallelism can be leveraged for low latency on different levels. The training of Gaussian Mixture Modes primarily requires matrix computation. If matrix computation is sped up by parallelism, more training can be run in the background at reduced wait times, resulting in both higher accuracy and lower latency. Also, giving models more iterations often leads them to converge with even less data, which also reduces latency. In the concrete example of diarization, lower runtime and therefore lower latency can be achieved by speeding up the cluster merge process, which might be parallelized on a thread level or using data parallelism by distributing each speaker model to a different core. With incoming data arriving through a sound card, USB device, or hard drive, I/O operations are likely to become a significant part of the runtime once parallelism is used intensively. Also, in the past we found that caching of highly repeated low-level operations (e.g., logarithm computations) helps runtime significantly. Therefore, a central cache for repeated operations seems highly desirable.

5 Conclusions

Automatic Speech Recognition (ASR) is an application that consistently benefits from more powerful computation platforms. With the increasing adoption of parallel multicore and manycore processors, we see significant opportunities for speech recognition in increasing recognition accuracy, increasing batch-recognition throughput, and reducing recognition latency. Here we have presented our on-going work on these directions, focusing on the opportunities and challenges with regard to parallelization. The proposed directions for future research may serve to guide future designs of efficient parallel software and hardware infrastructures for speech recognition.

Acknowledgments

This research is supported by Microsoft (Award #024263) and Intel (Award #024894) funding and by matching funding by U.C. Discovery (Award #DIG07-10227).

Notes

1. Mel-frequency cepstral coefficients (MFCCs) are a commonly used signal processing feature inspired by human hearing, which has higher resolution at low frequencies than at high frequencies.

References

[1] J. Ajmera and C. Wooters. A Robust Speaker Clustering Algorithm. In *In Proceedings of IEEE*

Workshop on Automatic Speech Recognition Understanding, pages 411–416, 2003.

[2] J. B. Allen. How do humans process and recognize speech? *IEEE Transactions on Speech and Audio Processing*, 2(4):567–577, October 1994.

[3] X. Anguera, C. Wooters, B. Peskin, and M. Aguilo. Robust speaker segmentation for meetings: The ICSI-SRI spring 2005 diarization system. In *Proceeding of the NIST MLMI Meeting Recognition Workshop*, Edinburgh, 2005. Springer.

[4] K. Asanovic, R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands, K. Keutzer, D. A. Patterson, W. L. Plishker, J. Shalf, S. W. Williams, and K. A. Yelick. The landscape of parallel computing research: A view from Berkeley. Technical Report UCB/EECS-2006-183, EECS Department, University of California, Berkeley, Dec 2006.

[5] H. Bourlard and S. Dupont. A new ASR approach based on independent processing and recombination of partial frequency bands. In *Proceedings of the International Conference on Speech and Language Processing (ICSLP)*, pages 426–429, Philadelphia, USA, October 1996.

[6] H. Bourlard, S. Dupont, and C. Ris. Multi-stream speech recognition. Technical Report RR 96-07, IDIAP research Institute, Martigny, Switzerland, December 1996.

[7] P. Cardinal, P. Dumouchel, G. Boulianne, and M. Comeau. GPU accelerated acoustic likelihood computations. In *Proc. Interspeech*, 2008.

[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 *Proceeding of the 10th Annual Conference of the International Speech Communication Association*, page 1183 1186, September 2009.

[9] D. Depireux, J. Simon, D. Klein, and S. Shamma. Spectro-temporal response field characterization with dynamic ripples in ferret primary auditory cortexa. *Journal of Neurophysiology*, 85(3), 2001.

[10] P. R. Dixon, T. Oonishi, and S. Furui. Fast acoustic computations using graphics processors. In *Proc. IEEE Intl. Conf. on Acoustics, Speech, and Signal Processing (ICASSP)*, Taipei, Taiwan, 2009.

[11] V. Gadde, A. Stolcke, D. Vergyri, J. Zheng, K. Sonmez, and A. Venkataraman. Building an ASR system for noisy environments: SRI's 2001 SPINE evaluation system. In *Proceedings of the International Conference on Speech and Language Processing (ICSLP)*, pages 1577–1580, Denver, USA, September 2002.

[12] D. Gelbart. Noisy numbers data and numbers speech recognizer. http://www.icsi.berkeley.edu/speech/papers/gelbart-ms/numbers.

[13] Y. Huang, O. Vinyals, G. Friedland, C. Müller, N. Mirghafori, and C. Wooters. A fast-match approach for robust, faster than real-time speaker diarization. In *Proceedings of the IEEE Automatic Speech Recognition Understanding Workshop*, 2007.

[14] S. Ishikawa, K. Yamabana, R. Isotani, and A. Okumura. Parallel LVCSR algorithm for cellphone-oriented multicore processors. In *Proc. IEEE Intl. Conf. on Acoustics, Speech, and Signal Processing (ICASSP)*, Toulouse, France, 2006.

[15] A. Janin, D. W. Ellis, and N. Morgan. Multistream: Ready for prime-time? In *Proceedings of the European Conference on Speech Communication and Technology (Eurospeech)*, volume 2, pages 591–594, Budapest, September 1999.

[16] D. Klein, D. Depireux, J. Simon, and S. Shamma. Robust spectrotemporal reverse correlation for the auditory system: optimizing stimulus design. *Journal of Computational Neuroscience*, 9(1):85–111, 2000.

[17] H. Ney and S. Ortmanns. Dynamic programming search for continuous speech recognition. *IEEE Signal Processing Magazine*, 16:64–83, 1999.

[18] ParLab web site. http://parlab.eecs.berkeley.edu.

[19] S. Phillips and A. Rogers. Parallel speech recognition. *Intl. Journal of Parallel Programming*, 27(4):257–288, 1999.

[20] M. Ravishankar. Parallel implementation of fast beam search for speaker-independent continuous speech recognition. Technical report, Computer Science and Automation, Indian Institute of Science, Bangalore, India, 1993.

[21] C. Wooters and M. Huijbregts. The ICSI RT07s Speaker Diarization System. In *Multimodal Technologies for Perception of Humans: International Ev aluation Workshops CLEAR 2007 and RT 2007*, pages 509–519, Baltimore, MD, USA, May 2007. Springer-Verlag.

[22] K. You, Y. Lee, and W. Sung. OpenMP-based parallel implementation of a continuous speech recognizer on a multi-core system. In *Proc. IEEE Intl. Conf. on Acoustics, Speech, and Signal Processing (ICASSP)*, Taipei, Taiwan, 2009.

[23] S. Zhao, S. Ravuri, and N. Morgan. Multi-stream to many-stream: Using spectro-temporal features

for asr. In *Proceedings of Interspeech*, Brighton, UK, September 2009.