Parallelizing Speaker-Attributed Speech Recognition for Meeting Browsing

Gerald Friedland, Jike Chong, Adam Janin
International Computer Science Institute
1947 Center Street, Suite 600
Berkeley, CA 94707
[fractor,jike,janin]@icsi.berkeley.edu

Abstract

The following article presents an application for browsing meeting recordings by speaker and keyword which we call the Meeting Diarist. The goal of the system is to enable browsing of the content with rich meta-data in a graphical user interface shortly after the end of meeting, even when the application runs on a contemporary laptop. We therefore developed novel parallel methods for speaker diarization and multi-hypothesis speech recognition that are optimized to run on multicore and manycore architectures. This paper presents the underlying parallel speaker diarization and speech recognition realizations, a comparison of results based on NIST RT07 evaluation data, and a description of the final application.

1 Introduction

Go to any meeting or lecture with the younger generation of researchers, business people, or government, and you will see a laptop or smartphone at every seat. Each laptop and smartphone is capable not only of recording and transmitting the meeting in real time, but also of advanced analytics such as speech recognition and speaker identification. These advanced analytics enable speech-based meta-data extraction that can be used to browse meeting recordings by speaker, keyword, and pre-defined acoustic events (e.g., laughter). The Meeting Diarist project aims to provide an interactive application running on your own laptop or smartphone that enables browsing, searching, and indexing of a meeting. Our target application is to provide an alternative to manual note-taking in multiparty meetings, providing additional functionality to what is available in the typical set of notes. In particular, since spoken language includes significant useful information besides the words, e.g., speaker identity and emotional affect (significantly cued by speaker intonation), enabling search through the complete audio record can be much more useful than a simple transcription.

The two major components of the Meeting Diarist are automatic speech recognition, which computes what was said, and speaker diarization, which computes who said it. The combination of the two allow users to search for relevant sections without having to review the entire meeting — for example, "What did the boss say about the timeline?" Both components are very expensive computationally. The best systems typically run on large clusters and are much slower than real-time (e.g., a one hour meeting might take 50 hours to process). By exploiting recent advances in parallel hardware and software, we can dramatically decrease the amount of time required to process a meeting. This article presents the main concepts used to parallelize state-of-the-art speaker diarization and speech recognition using both CPU and GPU parallelism.

2 Related Work

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 five PLUS processors with distributed memory in [14] with some success. The implementation statically mapped a carefully partitioned recognition network onto the multiprocessors, but the 3.8 $\times$ speed up was limited by runtime load imbalance, which would not scale to 30+ multiprocessors. The authors of [10] explored coarse-grained concurrency in large vocabulary conversational speech recognition (LVCSR) and implemented a pipeline of tasks on a cellphone-oriented multicore architecture. [19] proposed a parallel LVCSR implementation on a commodity multicore system using OpenMP. The Viterbi search in [19] was parallelized by statically partitioning a tree-lexical search network across cores. The parallel LVCSR system proposed in [13] uses a weighted finite state transducer (WFST) and data parallelism when traversing the recognition network. Prior work such as [7, 3] leveraged manycore processors and focused on speeding up the compute-intensive phase (i.e., observa-

978-0-7695-4217-1/10 $26.00 © 2010 IEEE
DOI 10.1109/ISM.2010.26

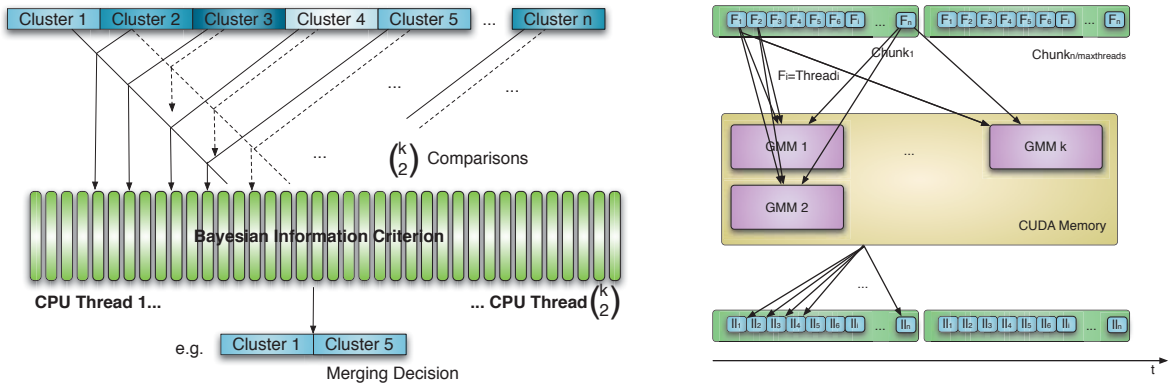

Figure 1. *Left: CPU parallelization of the merging calculation. Right: GPU parallelization of the log-likelihood calculation. Both are described in Section 3.2.*

tion probability computation) of LVCSR on manycore accelerators. Both [7] and [3] demonstrated approximately 5× 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.

Recently, some progress has been made on parallelizing the communication intensive phase (i.e., the Viterbi search). A complete data parallel LVCSR on the GPU with a LLM-based recognition network was presented in [6]. Parallel WFST-based LVCSR is also implemented on CPU and GPU in [18, 4]. [18] compared sequential and parallel implementations of the WFST-based recognition network representations. This paper contrasts the implications of using different recognition network representations on the GPU. In the following sections, we will briefly introduce the key differences in the recognition network representations as well as outline our implementation strategies to arrive at efficient parallel implementations.

Despite the initial successes of prior work in parallelizing speech recognition, at the time of writing this article, the authors were not able to find any prior work on the parallelization of a state-of-the-art speaker diarization system, with the exception of the authors' [8], which briefly described the application presented herein.

3 Speaker Diarization

The goal of speaker diarization is to segment a single or multi-channel audio recording into speaker-homogeneous regions with the goal of answering the question *who spoke when?* using virtually no prior knowledge of any kind (such as number of speakers, the words spoken, the language used, etc.). In practice, a speaker diarization system has to answer not just one, but two questions:

- What are the speech regions?
- Which speech regions belong to the same speaker?

Therefore, a speaker diarization system conceptually performs three tasks: First, discriminate between speech and non-speech regions; second, detect speaker changes to segment the audio data; and third, group the segmented regions together into speaker-homogeneous clusters. While this could in theory be achieved by a single clustering pass, in practice many speaker diarization systems use a speech activity detector as a first processing step and then perform speaker segmentation and clustering in one pass as a second step. Other pieces of information, such as the number of speakers in the recording, are extracted implicitly.

3.1 Approach

The following section outlines the traditional audio-only speaker diarization approach. We chose to parallelize the ICSI speaker diarization engine [17] as it performed very well in the 2007 and 2009 NIST Rich Transcription evaluations.

First, Wiener filtering [1] is performed on the audio channel for noise reduction. The HTK toolkit[1] is then used to convert the audio stream into 19-dimensional Mel-Frequency Cepstral Coefficients (MFCCs) [11], which are used as features for diarization. A frame period of 10 ms with an analysis window of 30 ms is used in the feature extraction. We use the same speech/non-speech segmentation as in [17], which is explained in [9]. It is an HMM/GMM approach originally trained on broadcast news data that generalizes well to meetings. These pre-processing steps are

[1] http://htk.eng.cam.ac.uk/

efficient and embarrassingly parallel as they can be concatenated in a producer/consumer pattern and processed in packets of 500 ms.

Then, in the actual segmentation and clustering stage of speaker diarization, an initial segmentation is generated by uniformly partitioning the audio track into k segments of the same length. k is chosen to be much larger than the assumed number of speakers in the audio track. For meeting data, we use $k = 16$.

The procedure for segmenting the audio data takes the following steps:

1. Train a set of GMMs for each initial cluster.
2. Re-segmentation: Run a Viterbi decoder using the current set of GMMs to segment the audio track.
3. Re-training: Retrain the models using the current segmentation as input.
4. Select the closest pair of clusters and merge them. At each iteration, the algorithm checks all possible pairs of clusters to see if there is an improvement in the Bayesian Information Criterion (BIC) [2] scores when the clusters are merged and the two models replaced by a new GMM trained on the merged cluster pair. The clusters from the pair with the largest improvement in BIC scores, if any, are merged and the new GMM is used. The algorithm then repeats from the re-segmentation step until there are no remaining pairs that when merged will lead to an improved BIC score.

A more detailed description can be found in [17].

The result of the algorithm consists of a segmentation of the audio track with k' clusters and an audio GMM for each cluster, where k' is assumed to be the number of speakers.

The output of a speaker diarization system consists of meta-data describing speech segments in terms of starting time, ending time, and speaker cluster label. This output is usually evaluated against manually-annotated ground truth segments. A dynamic programming procedure is used to find the optimal one-to-one mapping between the hypothesis and the ground truth segments so that the total overlap between the reference speaker and the corresponding mapped hypothesized speaker cluster is maximized. The difference is expressed as Diarization Error Rate, which is defined by NIST[2]. The Diarization Error Rate (DER) can be decomposed into two components: 1) Speech/non-speech error (speaker in reference, but non-speech in hypothesis, or speaker in hypothesis, but non-speech in reference), and 2) speaker errors (mapped reference is not the same as hypothesized speaker).The baseline single-distant microphone system as discussed here and presented in the NIST RT '07 evaluation, results in a DER of 21.03 %.

[2]http://www.itl.nist.gov/iad/mig/tests/rt/2003-spring/index.html

Component/Runtime	1 CPU	8 CPUs+GPU
Find & Merge Best Pair	0.372	0.04
Re-training/-alignment	0.168	0.01
Everything else	0.06	0.02
Total	0.6	0.07

Table 1. Runtime distribution of the ICSI Speaker Diarization System. Runtimes are given as $\times$ realtime, e.g., 0.1 means that 10 minutes of audio takes 1 minute to process.

3.2 Parallel Implementation

The goal of parallelizing speaker diarization is to increase the speed without harming the accuracy of the system. Using GPU parallelism has the advantage of exploiting the fine-grain parallel resources on these manycore systems. However, the cores are less powerful and implementation restrictions, such as the lack of I/O operations and operating system calls, makes it challenging to port code to a GPU. CPU parallelism on the other hand is easier to implement but there are significantly fewer cores and the current software solutions for implementing CPU parallelism do not allow for the same level of fine granularity as GPU tools. As a result, our solution is a hybrid, and two key implementation decisions resulted in the speedup:

1. Gaussian Mixture Model Training and BIC calculation is CPU parallelized. Each GMM is trained on a different CPU and each BIC comparison is performed in a different thread. Figure 1 (left side) illustrates the idea. The overall speed up is about a factor of five.
2. Calculation of the log-likelihoods is parallelized on the frame-level by creating one CUDA thread per frame. This resulted in near constant-time calculation of the log-likelihoods, since one core handles several threads concurrently. Practical experiments showed that the runtime is almost constant for up to 84,000 frames or 14 minutes of audio data. Figure 1 (right side) shows the idea.

The parallelized version of the speaker diarization system is as follows:

1. Train a set of GMMs for each initial cluster, each on a different CPU thread.
2. Re-segmentation: Run a Viterbi decoder using the current set of GMMs to segment the audio track. Log-likelihood computation is performed by transferring the GMMs onto a GPU and parallelizing on a frame-by-frame basis.

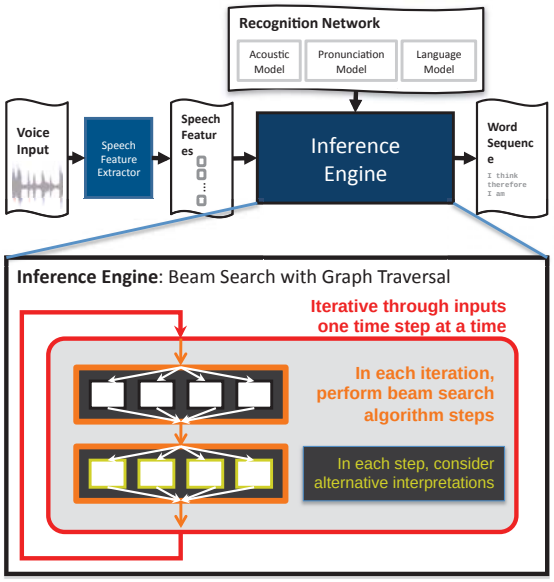

Figure 2. *Decoder Architecture as described in Section 4.*

3. Re-training: Retrain the models using the current segmentation as input. Each GMM is trained on a different CPU thread.

4. Select the closest pair of clusters and merge them. At each iteration, the algorithm checks all possible pairs of clusters to see if there is an improvement in the Bayesian Information Criterion (BIC) [2] scores when the clusters are merged and the two models replaced by a new GMM trained on the merged cluster pair. The clusters from the pair with the largest improvement in BIC scores, if any, are merged and the new GMM is used. The algorithm then repeats from the re-segmentation step until there are no remaining pairs that when merged will lead to an improved BIC score. Each merging decision is distributed to its on CPU thread.

We parallelized about 10k lines of code and brought the runtime from $0.6 \times$ realtime to $0.07 \times$ realtime on an 8-core Intel CPU with an NVIDIA GTX280 card without affecting the accuracy. Table 1 compares the runtimes of the different components.

4 Automatic Speech Recognition

Batch speech transcription is considered "embarrassingly parallel", a technical term used by the high performance computing community, if different speech utterances are distributed to different machines. However, there is significant value in improving computing efficiency, which is increasingly relevant in today's energy limited and form-factor limited devices and computing 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. Thus feature extraction is highly scalable with respect to the parallel platform advances. However, parallelizing the inference engine involves surmounting significant challenges.

Our inference engine traverses a graph-based recognition network based on the Viterbi search algorithm [12] 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 unpredictable memory accesses and poor data locality and cause significant challenges in load balancing and efficient synchronization between processor cores.

4.1 Parallelized Implementation

We implemented a data-parallel automatic speech recognition inference engine on the NVIDIA GTX480 graphics processing unit (GPU) by parallelizing over the inner most level of parallelism in the application that describe the thousands of alternative interpretations for conducting design space exploration. With the proper software architecture, our implementation has less than 16% sequential overhead, which promises more speedup on future, more parallel platforms [5]. Our parallelization of ASR involved three steps: description, architecting, and implementation.

In the description step, we exposed fine-grained parallelism by describing the operations of the ASR application. The algorithm structure of the inference engine is illustrated in Figure 2. The Hidden Markov model (HMM) based inference algorithm dictates that there be 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 step, where the inference engine keeps track of thousands to tens of thousands of alternative interpretations of the input waveform.

In the architecting step, we defined the design spaces to be explored. We made a design decision to implementing

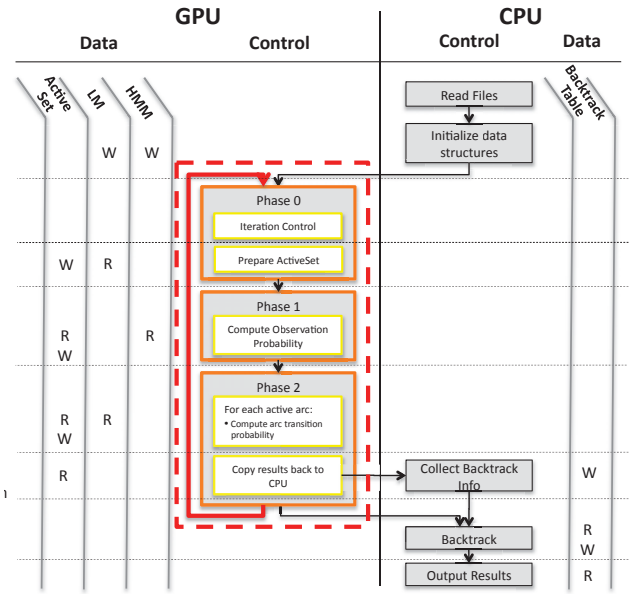

Figure 3. *Control and data flow between GPU and CPU (see Section 4).*

all parts of the Viterbi search algorithm on the GPU: Current GPUs' accelerator subsystems are controlled by a CPU over the PCI-Express data bus. With close to a TeraFLOP of computing capability on the GPUs, moving operands and results between CPU and GPU can quickly become a performance bottleneck. As shown in Figure 3, in the inference engine, there is a compute intensive phase (Phase 1) and a communication-intensive phase (Phase 2) of execution in each inference iteration. The computation-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 set of finite state machines representing the pronunciation and language models.

While we achieved 17.7 $\times$ speedup for the computation-intensive phase compared to sequential execution on the CPU, the communication-intensive phase is much more difficult to parallelize and received a 4.4 $\times$ speedup.

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 6.5-9.5$\times$ faster than realtime for the overall inference engine.

In the implementation step, we leveraged various optimization techniques to construct an efficient implementation. For highly parallel computing platforms such as the GPU, data regularity is very important for fully utilizing the available memory bandwidth to support the high compute throughput. We constructed runtime data buffers in each analysis time step to maximally regularize data access patterns. The data from the speech models required for inference 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.

For search and retrieval type applications, it is essential to provide not only the one-best hypothesis for the recognition results, but also the word lattice-based result with N-best hypotheses where less likely hypotheses of the utterance are also recorded. This allows for higher recall rate when a particular phrase of interest is requested by the user.

In our parallel implementation of the speech inference engine, the compute throughput of the implementation platform is being well-utilized such that the memory bandwidth bringing data to/from the processing elements becomes the speed-limiting resource for application execution. To support word lattice output, the top N-best path for each alternative interpretation must be recorded, where N is a parameter that can be set according to accuracy and execution speed trade-offs. Keeping track of N-best path leads to significant increases in the complexity of managing the inference process, as well as significant increases in memory bandwidth usage in recording N paths for backtracking. The performance implications are described in the next section.

4.2 Accuracy/Speed Evaluation

The speech models are taken from the SRI CALO real-time meeting recognition system [16]. The frontend uses 13-dimensional perceptual linear prediction (PLP) features with 1st, 2nd, and 3rd order differences, is vocal-track-length-normalized and is projected to 39 dimensions using heteroscedastic linear discriminant analysis (HLDA). The acoustic model is trained on conversational telephone and meeting speech corpora, using the discriminative minimum-phone-error (MPE) criterion. The language model is trained on meeting transcripts, conversational telephone speech, and web and broadcast data [15]. The acoustic model includes 52K triphone states which are clustered into 2,613 mixtures of 128 Gaussian components. The pronunciation model contains 59K words with a total of 80K pronunciations. We use a small back-off bigram language model with 167k bigram transitions. The decoding algorithm is linear lexicon search.

The test set consisted of excerpts from NIST conference meetings taken from the "individual head-mounted micro-

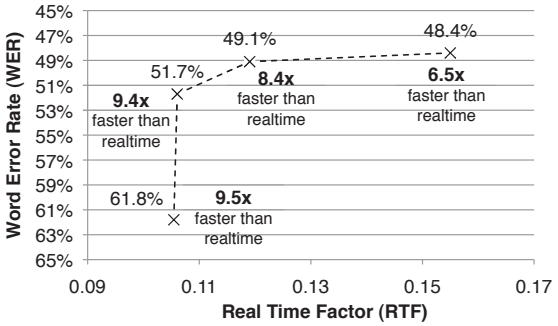

Figure 4. *Accuracy vs speed graph of the parallel speech decoder*

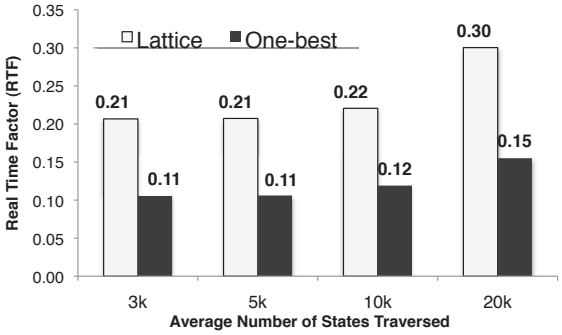

Figure 5. *Speedup the parallel speech decoder while generating one-best path or lattice output*

phone" condition of the 2007 NIST Rich Transcription evaluation. The segmented audio files total 3.1 hours in length and comprise 35 speakers. The meeting recognition task is very challenging due to the spontaneous nature of the speech[3]. The ambiguities in the sentences require larger number of active states to keep track of alternative interpretations which leads to slower recognition speed.

Our recognizer uses an adaptive heuristic to adjust the search beam size based on the number of active states. It controls the number of active states to be below a threshold in order to guarantee that all traversal data fits within a pre-allocated memory space. Figure 4 shows the decoding accuracy, with varying thresholds and the corresponding decoding speed on various platforms. The accuracy is measured by word error rate (WER), which is the sum of substitution, insertion and deletion errors. Recognition speed is represented by the real-time factor (RTF) which is computed as the total decoding time divided by the duration of the input speech. As shown in Figure 4, the parallel implementations can achieve very fast recognition speed.

For the experimental manycore platform, we use a Intel Core i7 Quadcore-based host system with 12GB host memory and a GTX480 graphics card with 1.5GB of device memory. We analyze the performance of our inference engine implementations on the GTX480 manycore processor.

The lattice performance results are shown in Figure 5. The X-axis shows the average number of states traversed, where each state is an alternative interpretation of the utterance tracked at each time step. An adaptive beam-threshold algorithm is used to keep the number of states around a particular level allocated for the inference process. The Y-axis shows the speed of the implementation as a real time factor (RTF), which measures the number of seconds of processing required for each second of speech input data.

We chose an operating point where the total execution time takes twice as long to build a lattice compared to tracking just the one-best path. With this operating point, we were able to maintain 9 top paths for each alternative interpretations. Compared to tracking the one-best path, the execution time of the key kernels in Phase 2 of the algorithm increased by $3\times$, and the memory transfer time moving backtrack information from GPU device to CPU host increased by $5\times$.

[3]A single-pass time-synchronous Viterbi decoder from SRI using lexical tree search achieves 37.9% WER on this test set

5 The Application

Figure 6 shows a current version of the Meeting Diarist. The speaker diarization output and the speech recognition output are combined with the audio file into a GUI. We expect the Meeting Diarist to replace a typical YouTube video/audio player. The browser shows either a video or hand-selected images of the speakers and allows play and pause, as well as seeking to random positions. The navigation panel on the bottom shows iconized frames of a video or the speaker images. It allows a user to directly jump to the beginning time of a dialog element. When acoustic event detection is used in addition, these events can serve as segment boundaries for further navigation elements, e.g., laughter for funny remarks. Also, the current dialog element is highlighted while the show is playing. In order to make navigation more selective, the user can deselect one or more speakers. The user can also search for dialog elements by keyword. This is performed by the traversing the hypothesis lattice of the speech recognizer and matching it with the entered keyword.

6 Conclusion and Future Work

This article presents a multi-hypothesis speech-based Meeting Diarist and discusses how low-latency meeting analysis is made possible through novel hybrid CPU and GPU parallelization strategies for speaker diarization and

speech recognition. Speech Recognition and Diarization are applications that consistently benefit from more powerful computation platforms. With the increasing adoption of parallel multicore and manycore processors, we see significant opportunities for increasing recognition accuracy, increasing batch-recognition throughput, and reducing recognition latency. The faster processing will also open more possibilities for the further incorporation of data e.g., allowing multimodal approaches. This articles has presented our on-going work on these directions, focusing on the opportunities and challenges for parallelization. Future work will include the following:

For the inference engine component, there is an interesting trade-off that can be explored to more efficiently track N-best path for search and retrieval applications. Currently in Phase 2 of the algorithm, transferring the top 9 paths from GPU to CPU for all active states in every time step causes a 5× increase in the data transfer time as compared to tracking only the one-best paths. By caching active states across multiple time steps and deferring the data transfer, one can detect whether the states are still part of the N-best paths being tracked a few time steps later. The hypothesis is that many states can be eliminated before having to transfer their lattice information from the GPU to the CPU. This would reduce the memory transfer time, at the expense of increased computation on the GPU. As the memory transfer time is a sequential process and the computation is a parallel process, and as the GPUs become more parallel, this should be a beneficial optimization in the long run.

The parallelization of the speaker diarization component is work in progress. Parallelism can be leveraged for low-latency on different levels. The training of Gaussian Mixture Models, for example, 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. The Viterbi alignment component could be sped up by a similar implementation as in the speech recognition component. Finally, evaluating the system with a real user study would give hints on where accuracy has to be improved further.

7 Acknowledgments

This research is supported in part by an Intel Ph.D. Research Fellowship, by Microsoft (Award # 024263) and Intel (Award # 024894) funding and by matching funding by U.C. Discovery (Award # DIG07-10227).

Figure 6. The interface of the Meeting Diarist.

References

[1] A. Adami, L. Burget, S. Dupont, H. Garudadri, F. Grezl, H. Hermansky, P. Jain, S. Kajarekar, N. Morgan, and S. Sivadas. Qualcomm-icsi-ogi features for asr. In *Proc. ICSLP*, pages 4–7, 2002.

[2] J. Ajmera and C. Wooters. A robust speaker clustering algorithm. *Automatic Speech Recognition and Understanding, 2003. ASRU'03. 2003 IEEE Workshop on*, pages 411–416, 2003.

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

[4] 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. *10th Annual Conference of the International Speech Communication Association (InterSpeech)*, September 2009.

[5] J. Chong, E. Gonina, K. You, and K. Keutzer. Exploring recognition network representations for efficient speech inference on highly parallel platforms. In *11th Annual Conference of the International Speech Communication Association (InterSpeech)*, 2010.

[6] J. Chong, Y. Yi, A. Faria, N. Satish, and K. Keutzer. Data-parallel large vocabulary continuous speech recognition on graphics processors. *Proceedings of the 1st Annual Workshop on Emerging Applications and Many Core Architecture*, pages 23–35, June 2008.

[7] 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.

[8] G. Friedland, J. Chong, A. Janin, and C. Oei. A parallel Meeting Diarist. *Proceedings of the SSSC 2010 Workshop at ACM Multimedia*, to appear, 2010.

[9] M. Huijbregts. *Segmentation, Diarization, and Speech Transcription: Surprise Data Unraveled*. PrintPartners Ipskamp, Enschede, The Netherlands, 2008.

[10] 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.

[11] P. Mermelstein. Distance measures for speech recognition, psychological and instrumental. *Pattern Recognition and Artificial Intelligence*, pages 374–388, 1976.

[12] H. Ney and S. Ortmanns. Dynamic programming search for continuous speech recognition. *IEEE Signal Processing Magazine*, 16, 1999.

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

[14] M. Ravishankar. Parallel implementation of fast beam search for speaker-independent continuous speech recognition, 1993.

[15] A. Stolcke, X. Anguera, K. Boakye, O. Cetin, A. Janin, M. Magimai-Doss, C. Wooters, and J. Zheng. The SRI-ICSI Spring 2007 meeting and lecture recognition system. *Lecture Notes in Computer Science*, 4625(2):450–463, 2008.

[16] G. Tur et al. The CALO meeting speech recognition and understanding system. In *Proc. IEEE Spoken Language Technology Workshop*, 2008.

[17] C. Wooters and M. Huijbregts. The ICSI RT07s speaker diarization system. In *Proceedings of the Rich Transcription 2007 Meeting Recognition Evaluation Workshop*, 2007.

[18] K. You, J. Chong, Y. Yi, E. Gonina, C. Hughes, Y. Chen, W. Sung, and K. Keutzer. Parallel scalability in speech recognition: Inference engine in large voc abulary continuous speech recognition. (6):124–135, November 2009.

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