A Parallel Meeting Diarist

Gerald Friedland
International Computer Science Institute
1947 Center Street, Suite 600
Berkeley, CA 94704-1198
fractor@icsi.berkeley.edu

Jike Chong
UC Berkeley
Dept. of EECS
Berkeley, CA 94720
jike@berkeley.edu

Adam Janin
International Computer Science Institute
1947 Center Street, Suite 600
Berkeley, CA 94704-1198
janin@icsi.berkeley.edu

ABSTRACT

The following article presents an application for browsing meeting recordings by speaker, keyword, and pre-defined acoustic events (e.g., laughter), 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 (GUI) 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 speech recognition that are optimized to run on multicore and manycore architectures. This paper presents the application and the underlying parallel speaker diarization and speech recognition realizations.

Categories and Subject Descriptors

H5.5 [**Information Interfaces and Presentation**]: Sound and Music Computing—*Signal analysis, synthesis, and processing*; H5.4 [**Information Systems Applications**]: Navigation

General Terms

Experimentation

Keywords

parallel computing, speaker diarization, speech recognition, video navigation, meetings, rich transcription

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 metadata 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. Figure 1 shows the overall architecture of the system. 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 [7] 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 [5] explored coarse-grained concurrency in large vocabulary conversational speech recognition (LVCSR) and implemented a pipeline of tasks on a cellphone-oriented multicore architecture. [10] proposed a parallel LVCSR implementation on a commodity multicore system using OpenMP. The Viterbi search in [10] was parallelized by statically partitioning a tree-lexical search network across cores. The parallel LVCSR system proposed in [6] uses a weighted finite state transducer (WFST) and data parallelism when traversing

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SSCS'10, October 29, 2010, Firenze, Italy.
Copyright 2010 ACM 978-1-4503-0162-6/10/10 ...$10.00.

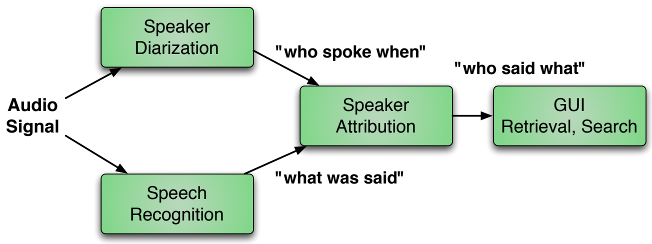

Figure 1: Overview diagram of the meeting diarist.

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 $\times$ real-time, e.g. 0.1 means that 10 minutes audio take 1 minute of processing.

the recognition network. Prior work such as [4, 1] leveraged manycore processors and focused on speeding up the compute-intensive phase (i.e., observation probability computation) of LVCSR on manycore accelerators. Both [4] and [1] demonstrated approximately 5$\times$ 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. A complete data parallel LVCSR on the GPU with a LLM-based recognition network was presented in [3]. Parallel WFST-based LVCSR is also implemented on CPU and GPU in [9, 2]. [9] 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.

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.

We chose to parallelize the ICSI speaker diarization engine [8].

The result of the algorithm consist 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 name. 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[1]. 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 Speaker Diarization System used for these experiments has competed in the NIST evaluations of the past several years and established itself well among state-of-the-art systems[2]. The baseline single-distant microphone system as discussed here and presented in the NIST RT '07 evaluation, results in a DER of 21.03 %.

3.1 Parallel Implementation

The goal for parallelizing speaker diarization was to increase the speed without harming the accuracy of the system. We parallelized about 10k lines of code and brought down 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. Using GPU parallelism has the advantage of being able to use fine-grain parallel resources on many cores. However, the cores are less powerful and implementation restrictions, such as the lack of IO operations and operating system calls, making 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. Therefore our solution is a hybrid and with two key implementation decisions that resulted in the speedup:

[1]http://www.itl.nist.gov/iad/mig/tests/rt/2003-spring/index.html
[2]NIST rules prohibit publication of any rankings. Please refer to the NIST website for further information: http://www.itl.nist.gov/iad/mig/tests/rt/

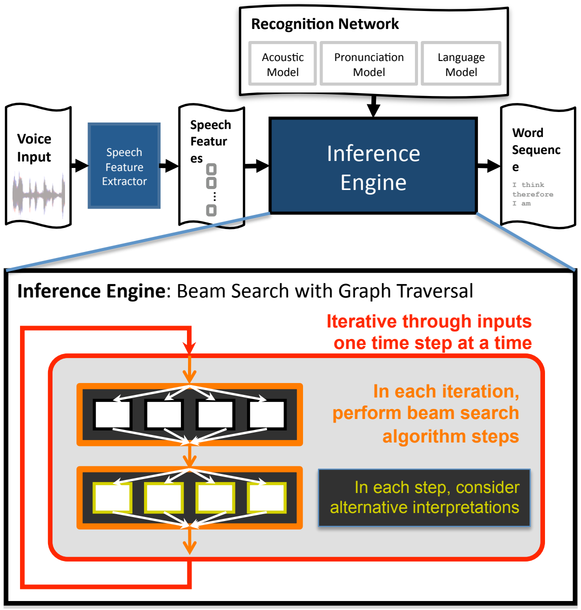

Figure 2: Decoder Architecture as described in Section 4.

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. 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 NVidia CUDA (Compute Unified Device Architecture) 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. Table 1 compares the runtimes of the different components.

4. AUTOMATIC SPEECH RECOGNITION

We implemented a data-parallel automatic speech recognition inference engine on the NVIDIA GTX280 graphics processing unit (GPU), and achieved over 10 × speedup compared to single instruction, multiple data (SIMD) optimized sequential implementation on a single core on an Intel core i7 CPU. It was important to expose 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 8 % sequential overhead, which promises more speedup on future, more parallel platforms [9]. 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.

Avg. # of Active States		32820	20000	10139	3518
WER		41.6	41.8	42.2	44.5
RTF	Sequential	4.36	3.17	2.29	1.20
	Multicore	1.23	0.93	0.70	0.39
	Manycore	0.40	0.30	0.23	0.18

Table 2: Accuracy, word error rate (WER), for various beam sizes and corresponding decoding speed in real-time factor (RTF)

In the architecting step, we defined the design spaces to be explored. We made a design decision to implementing all parts of the Viterbi search algorithm on the GPU: Current GPUs' accelerator subsystems are controlled by a CPU over the PCIe 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. In the inference engine, there is a compute intensive phase and a communication-intensive phase 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 finite state transducer representing the pronunciation and language models. While we achieved 17.7 × 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 × 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 11.3 × speedup of the overall inference engine.

In the implementation step, we leveraged various hardware and system support infrastructure to construct efficient implementations. The two most important implementation optimizations were:

1. Leveraging fast hardware atomic operation support: The inference process is composed of data-parallel graph traversals on the recognition network. The graph traversal routines execute 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.

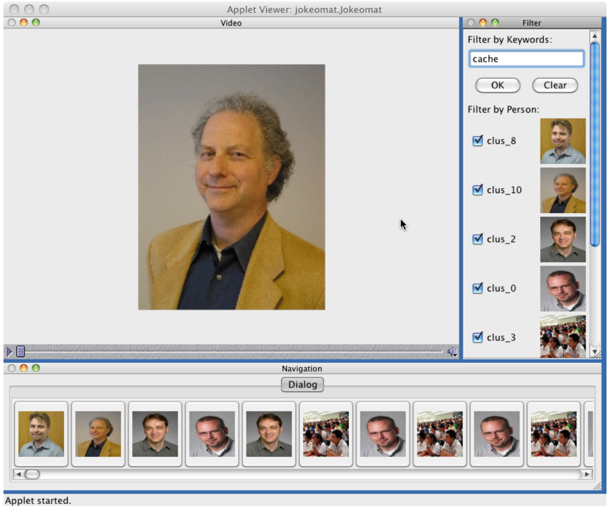

Figure 3: Interface of the Meeting Diarist.

2. Constructing 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.

5. THE APPLICATION

Figure 3 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 either a dialog element. When acoustic event detection is used in addition, these events can serve as segement 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 and select dialog elements by keyword as determined by the speech recognizer.

6. CONCLUSIONS AND FUTURE WORK

This article presents a 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. Parallelism can be leveraged for fast-response (low-latency) on different levels, especially in speaker diarization. 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. 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 for parallelization.

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).

7. REFERENCES

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

[2] 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.

[3] 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.

[4] P. R. Dixon, T. Oonishi, and S. Furui. Fast acoustic computations using graphics processors. In *IEEE ICASSP*, Taipei, Taiwan, 2009.

[5] S. Ishikawa, K. Yamabana, R. Isotani, and A. Okumura. Parallel LVCSR algorithm for cellphone-oriented multicore processors. In *IEEE ICASSP*, Toulouse, France, 2006.

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

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

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

[9] 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.

[10] K. You, Y. Lee, and W. Sung. OpenMP-based parallel implementation of a continous speech recognizer on a multi-core system. In *IEEE ICASSP*, Taipei, Taiwan, 2009.