A High-Performance Hardware Speech Recognition System for Mobile Applications

Patrick Bourke, Rob A. Rutenbar
{pbourke, rutenbar}@ece.cmu.edu

ECE Department
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

While commercial speech recognition systems remain limited in their capabilities, research systems are now relatively mature, although computationally expensive. Specialized hardware is one solution to this problem, and certainly the only solution at present for mobile applications. We present a queue-based hardware architecture for large-vocabulary, speaker-independent, continuous, real-time speech recognition in the mobile environment, demonstrating better than real-time performance. We base our results on simulation of approximately one hour of speech data for a 5,000 word vocabulary.

1 Introduction

Speech recognition is a technology that can trace its roots back to research programs initiated in the early 1970's, yet only recently have we started to see widespread use of speech recognition in commercial applications. Prime examples include voice dialing on cell phones, automated call-centers, and desktop software packages such as IBM's ViaVoice [1] and Dragon's Naturally Speaking [2]. Such applications, however, are still extremely limited in their abilities; we are far from realizing the simple and ubiquitous human-machine interface speech recognition seems to promise.

By comparison with commercial recognition systems, however, the very best research systems are now approaching their ultimate goal—large-vocabulary, continuous, speaker-independent, real-time recognition. These systems are large-vocabulary in that they can handle on the order of 60,000 words; continuous in that they recognize natural human speech, spoken without deliberate pauses; and speaker-independent in that no user-specific training is required. Typical accuracy rates may be as high as 90% [3].

Unfortunately, these systems are also extremely computationally intensive, requiring the full processing resources of a modern desktop machine in order to run in real-time. This completely rules out high-quality speech recognition for many of the applications where one might want it most—in particular, for mobile applications.

One solution to this problem is application-specific integrated circuits (ASICs), which are ideally suited to tasks too performance- or power-hungry for general purpose processors. Given both the computational demands and potential applications, high-performance speech recognition is an excellent candidate for this treatment. Accordingly, this paper proposes a high-performance hardware speech recognition system designed specifically for mobile applications. In particular, we present a novel queue-based memory architecture to (1) address the need in modern speech recognition systems for highly irregular access to extremely large data sets, and (2) permit use of a flash-based memory system well suited to mobile applications.

We base our hardware speech recognition system on the Sphinx 3.0 software recognizer developed at Carnegie Mellon University [3,4]. The Sphinx system is one of the premier large-vocabulary, continuous, speaker-independent research recognition systems in the world today. Sphinx describes speech using *Hidden Markov Models* (HMMs), a widely used and extremely successful speech recognition technique [5].

The organization of this paper is as follows. In the following section we give an overview of the Sphinx 3.0 software recognizer on which we base our hardware architecture. In Section 3 we present this architecture. Section 4 discusses simulation results characterizing the architecture, while in Section 5 we suggest some directions for future research and make some concluding remarks.

2 The Sphinx 3.0 Speech Recognizer

The Sphinx 3.0 software recognizer on which we base our hardware architecture recognizes speech using Hidden Markov Models (HMMs). Hidden Markov Models, just like ordinary Markov Models, are simply sets of states linked by transitions with some probability. For instance, a Markov Model for a coin toss experiment might have two states, heads and tails, with equal probabilities for moving to each state (if the coin is unbiased). Hidden Markov Models differ simply in that each state does not represent some experimental outcome (such as heads or tails), but rather such outcomes are generated according to a *probability distribution* associated with each state. Unlike an ordinary Markov model, therefore, when we observe heads or tails, we cannot be sure in which state a Hidden Markov Model is—hence the state of the model is said to be "hidden".

As applied to the problem of speech recognition, the observations made are suitably-processed human speech, and speech recognition is achieved by finding the series of Hidden Markov Model states that most likely generated those observations. The algorithm for doing this is *Viterbi search*, which calculates the probabilities for transitions to all future HMM states from some present state and, through suitable record-keeping, allows one to recover the series of transitions from initial state to the most likely final state [6]. As every state may transition to many future states, for a complex speech recognition system the Viterbi search space will rapidly become unmanageable. Certain low probability forward transitions are therefore ignored, a technique known as *pruning*.

At the most fundamental level, Sphinx models speech as a series of unique sounds, or *phones*, corresponding to different positions of features in the mouth and nasal passages during speech. In the English language, there are about 50 phones. Examples would be the "a" sound in "apple" or the "th" sound in "the". Additionally, the sound of a phone is influenced by the phones preceding and following it, as the anatomical features responsible for human speech smoothly transition between different positions. Phones are therefore assembled into *triphones*, or triplets of phones, to model speech. It is these triphones that are represented by Hidden Markov Models in Sphinx, which are then further assembled to form higher level constructs, such as words and chains of words. Each Sphinx triphone HMM consists of four sequential states, with self-transitions for the first three states.

Sphinx processes speech in 10ms time intervals, referred to as *frames*, and processing may be divided into two distinct stages, a *front-end* and a *back-end*. The front-end takes speech

input, performs signal processing to calculate certain quantities representative of a particular speech sample (called *features*), and passes these through a soft-boundary classification technique known as a *Gaussian Mixture Model* (GMM). The outputs of the GMM are scores for how well every HMM state matches the speech input for a frame. For practical purposes, however, individual scores for each state are not determined, but rather states are clustered for scoring into groups called *senones*. The clustered scores are therefore *senone scores* [7].

The back-end performs speech recognition proper, running the Viterbi algorithm using the senone scores provided by the front-end. To achieve high accuracy, however, actual recognition is far more sophisticated than simply performing Viterbi. In fact, what actually occurs may be thought of as Viterbi search operating on three levels simultaneously—processing the transitions within triphone HMMs, between triphone HMMs, and finally between words composed of these HMMs. This is illustrated in Figure 1.

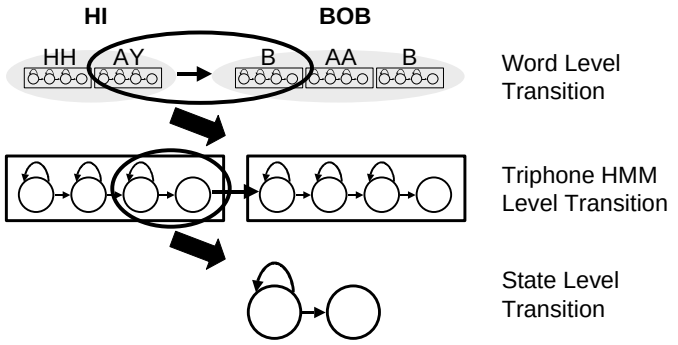

Figure 1: Viterbi search operating on multiple levels

While within-triphone transitions are reasonably straightforward, those between triphones and between words are not. Transitioning to new triphones is based on sequences of triphones corresponding to every possible word that might be recognized, while transitions between words are governed by a complex *language model*. This language model accounts for the relative probabilities of recognizing each word alone (*unigram* probabilities), plus certain pairs (*bigrams*) and triples (*trigrams*) of words. It may be hundreds of megabytes in size for large vocabularies.

The large data sets accessed by Sphinx during recognition may be broadly divided into two categories: *dynamic data* is both read and written when recognizing a frame of speech, while *static data* is read only. Dynamic data consists of currently evaluating, or *active* HMM data (i.e., the Viterbi search space for the current frame) and a list of recognized words. Static data consists of all pre-determined speech model data relating phones, triphones, words, unigrams, bigrams, trigrams, and so on. With the exception of a few small, extremely frequently accessed memories, we assume both dynamic and static data are predominantly stored in flash memory, used commonly in mobile applications.

3 Architecture

With the Sphinx speech recognizer introduced, we now discuss our proposed hardware architecture for high-performance, mobile speech recognition. Simulation results characterizing the architecture are provided in Section 4.

3.1 Chip Level Architecture

The division between front-end and back-end processing in Sphinx is maintained in our hardware architecture. As the front-end consists primarily of straightforward signal processing, for which many hardware performance improvement techniques exist, we concentrate on the back-end architecture for the remainder of this paper. The architecture we envisage at the chip level is illustrated in Figure 2. A hardware front-end performs the DSP functions necessary to provide senone scores to the hardware back-end, which performs speech recognition proper.

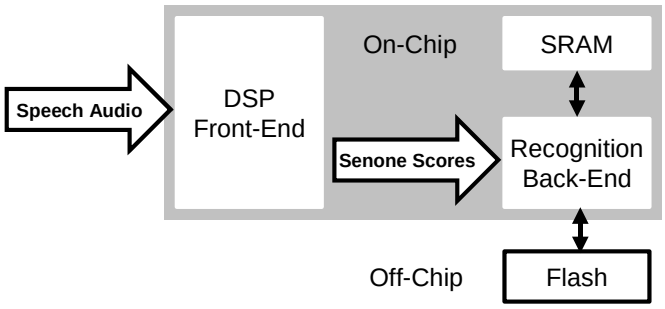

Figure 2: Chip-level architecture

3.2 Back-End Architecture

We identify three major processing stages in the back-end recognition process—calculating Viterbi state probabilities (scoring), transitioning between HMMs, and evaluating the complex language model—the latter being most computationally intensive. These functions are handled by three hardware engines—the *Scoring Engine*, *Transition Engine* and *Language Model Engine* respectively. This architecture is illustrated in Figure 3.

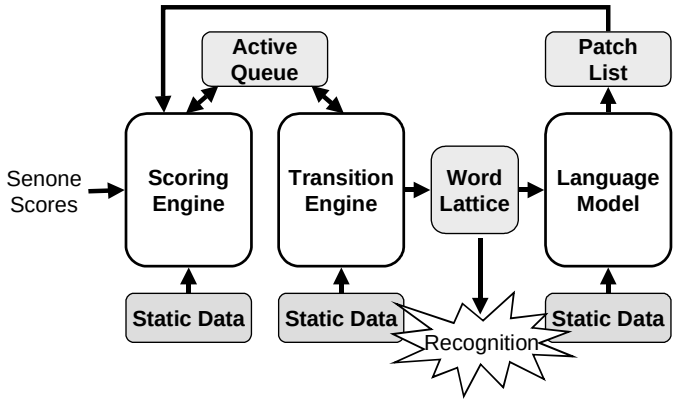

Figure 3: Back-end architecture

The three hardware engines are in turn linked via three memories that hold all dynamic data necessary for recognition—these are the *Active Queue*, the *Word Lattice* and the *Patch List*. The Active Queue contains an entry for every triphone HMM being considered in the current speech frame (i.e., the current Viterbi search space). Its contents are processed sequentially and updated by the Scoring and Transition Engines every frame, yielding the Viterbi search space for the following frame and any recognized words. Recognized word data is stored in the Word Lattice.

Based on the contents of the Word Lattice, the Language Model identifies new words that may potentially be recognized and adds them to the Viterbi search space for the next frame. Rather than directly modifying the Active Queue, this data is stored to the Patch List memory, to be "patched" into the Active Queue by the Scoring Engine in the following frame. Decoupling the Language Model in this way allows the Active Queue to remain a sequentially accessed structure.

In addition to the three dynamic memories, each processing engine is also fed by memories storing the static data necessary for their respective computations.

In greater detail, processing proceeds through the three engines as follows:

1. Each HMM in the Active Queue passes through the Scoring Engine and the Viterbi probability calculation is performed, assigning all HMMs new scores. For some HMMs occupying the start of a word, the Patch List will be consulted during scoring.

2. The Transition Engine reads in newly scored HMMs from the Active Queue, dropping those not meeting a pruning threshold. Of the remaining HMMs, the action taken depends on the HMM's position within a word. HMMs occupying the last position within a word are checked against a further threshold to determine if that word scores highly enough to be recognized. If so, the word is entered into the Word Lattice. For non-final HMMs, additional HMMs are added to the queue according to what sound should follow next for each word.

3. The Language Model examines the newly recognized words in the current frame and their predecessors in earlier frames. Based on the existence and probabilities of trigrams, bigrams and unigrams, possible follower words are identified. Data describing the initial HMM for these followers is entered into the Patch List.

Active HMM data thus cycles through the Scoring and Transition Engines once per frame, with feedback from the Language Model, to produce a list of recognized words in the Word Lattice.

A further difficulty of the Sphinx back-end algorithm is that certain parts of the algorithm must be performed serially. In particular, while it is possible for the Transition Engine and Language Model to run in parallel, the Scoring Engine must complete before any further processing can take place. It is therefore desirable that the Scoring Engine run as quickly as possible at the start of each frame. For power reasons, it may then be shut down until the next frame. We discuss the processing breakdown between the three engines in Section 4.

3.3 A Mobile Memory Model

While functionally identical, flow of the original Sphinx algorithm has been modified in our hardware implementation to enable active HMM data to be queued in the Active List (at the cost of some complexity). This modification has two significant advantages:

- *Prefetching of Static Data.* As the Scoring Engine processes every entry in the Active Queue sequentially, it is possible to look ahead to the next entry and prefetch the static data needed, thereby reducing memory stall time.

- *Compatibility with a flash memory model.* Since the Active Queue is only ever accessed sequentially, it is possible to take full advantage of burst access modes provided by memory types such as flash, to rapidly read this structure.

These two advantages are particularly important in improving the performance of the Scoring Engine which, as discussed above, lies on the critical processing path, and requires significant bandwidth to the Active Queue.

Prior to discussing simulation results, we now present specific details of the memory model. As previously stated, a flash-based memory system is employed, similar to those found in mobile (e.g. cell phone) applications today, in which flash memory is not only used as storage, but also as main memory (known as *Execute-in-Place*). Memory bus widths and sizes are illustrated in Figure 4.

Flash memory is used almost exclusively for all dynamic and static memories, with the following exceptions. Firstly, the Scoring Engine uses 90 kilobytes of on-chip SRAM to store

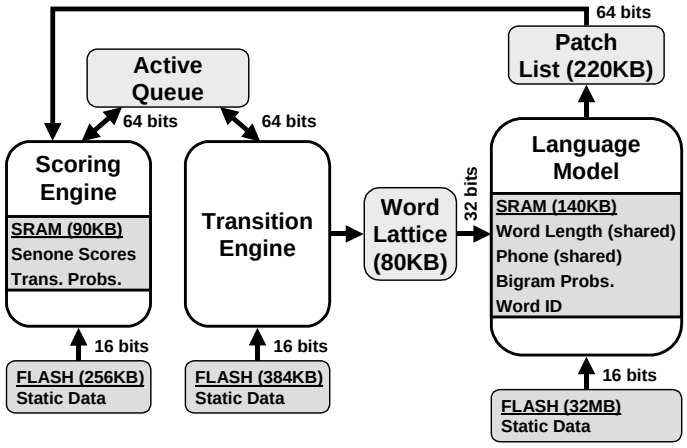

Figure 4: Memory sizes and bus widths

incoming senone scores and the transition probabilities for all HMMs. Secondly, 140 kilobytes of on-chip SRAM is used to store the four static data structures most frequently accessed by the Language Model (two of which are shared with the Scoring and Transition Engines). These are the number of triphones per word (Word Length), the phones comprising each word (Phone), possible bigram probability values (Bigram Probs.) and a unigram index (Word ID).

All remaining static data is stored in off-chip flash, the Language Model requiring the largest static memory at 32 megabytes, while the Scoring and Transition Engines require 256 kilobytes and 384 kilobytes respectively. Each processing engine accesses static data stored in flash across a 16-bit bus.

In terms of dynamic memory, Word Lattice and Patch List memory sizes are 80 kilobytes and 220 kilobytes respectively; Active Queue size depends on the speech being recognized and is discussed in the following section. The Word Lattice is accessed across a 32-bit bus, while Active Queue and Patch List accesses are 64 bits wide. Since the Scoring and Transition Engines do not run simultaneously, they can in practice share static memory and Active Queue busses.

We assume flash memory with a random access time of 62.5ns and a burst access time of 12.5ns, running at a clock frequency of 80 MHz [8,9].

4 Simulation and Results

In this section we evaluate the memory behavior and performance of our hardware speech recognition architecture. We examine Active Queue behavior and determine for a medium sized vocabulary the necessary capacity of this structure. Finally, static flash memory bandwidths and real-time performance are assessed.

We evaluate the architecture using event-driven simulation, for a medium sized vocabulary model of approximately 5,000 words. While not the largest possible language model, this is nonetheless a speech recognition task of considerable complexity that, at present, still requires the processing power of a modern desktop computer to run. Approximately one hour of speech was simulated, or just under 400,000 frames. The parameters for this model are listed in Table 1.

Table 1: 5,000 word language model parameters

Words	5,000
Phones	49
Triphones	110,879
Unigrams	4,989
Bigrams	1,639,687
Trigrams	2,684,151

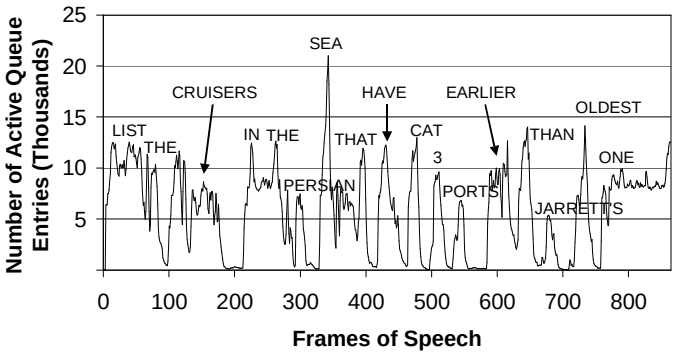

Figure 5: Active Queue size across 8.5 seconds of speech

Turning our attention to the Active Queue, Figure 5 shows the number of active HMM entries in this structure across 850 typical frames of speech, or 8.5 seconds. The number of active entries, which is proportional to processing intensity, varies wildly depending on the speech being recognized. Interestingly, this data also still correlates with the original speech. The speech sample of Figure 5 represents the sentence *"List the cruisers in the Persian Sea that have CAT-3 ports earlier than Jarrett's oldest one"*—each peak in Active Queue activity is linked on the plot to what was said. For a full hour of speech, the distribution of Active Queue sizes is shown in Figure 6. During speech, Active Queue size remains around 12,000 entries, while the large number of frames with very few active entries are accounted for by pauses between words. For entries that are 32 bytes, a single megabyte of flash memory is capable of storing 99.99% of all occurring Active Queues. Were the Active Queue ever to overflow, entries could simply be dropped, with the effect of slowly degrading recognition accuracy. Active Queue bandwidth is proportional to the number of entries, on average 30.8 megabytes/sec.

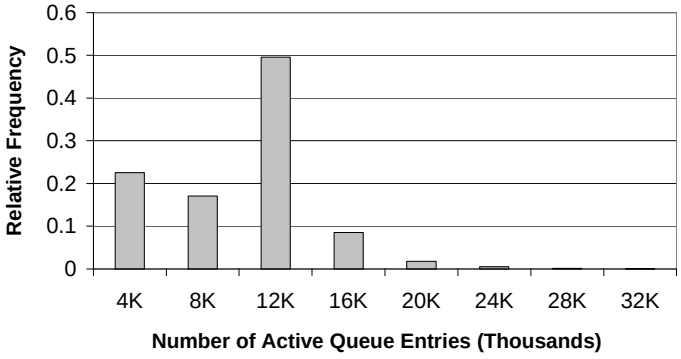

Figure 6: Distribution of Active Queue sizes

In each processing engine, static memory bandwidths for both SRAM and flash accesses are shown in Figure 7. SRAM accesses are shaded in gray. Flash memory accesses are broken down into number of right context phones, left context phone index and left phone ID—all used to identify a single phone within a triphone—then bigram and all other accesses. We omit Scoring Engine flash and phone accesses, which are negligible.

It is clear from Figure 7 why the Language Model Engine dominates processing—Language Model flash memory accesses, predominantly to bigram data, are double that to any other static memory, including those small enough to reside in SRAM. While similar bandwidth numbers are seen for the Active Queue, Language Model static accesses are not able to take advantage of flash burst modes to anywhere near the degree of the serially accessed Active Queue.

On average, a single frame of speech takes 374,000 cycles to process. At a clock rate of 80 MHz, this translates into recognition at 0.47 X real-time, or about twice the speed of real-time recognition. Language Model processing accounts for 80.1% of this, the Scoring Engine 19.9%. The Transition Engine completes in 42% of the time taken by the Language Model with which it runs in parallel.

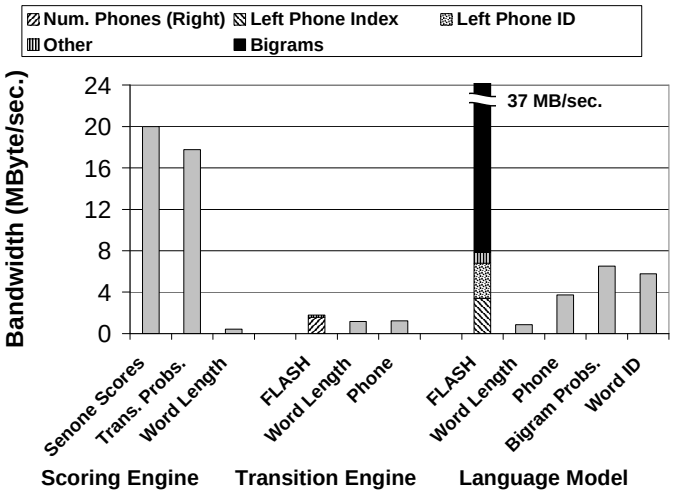

Figure 7: Processing engine static memory bandwidths

The primary factor influencing Sphinx software performance is ability to access large data sets at high speed, and in particular Language Model static data. As expected then, we observe that the twice real-time recognition rate achieved in hardware is also limited by Language Model static memory bandwidth.

5 Future Work and Conclusions

For mobile applications, power is obviously a key consideration, and a thorough power analysis remains to be done for this work. For inclusion in a cell phone, for instance, a power budget of less than 100mW would be necessary. Other research directions include exploring more complex memory hierarchies to improve performance, different or larger vocabulary language models and, of course, building actual hardware.

In this paper we have presented a hardware architecture for large-vocabulary, speaker-independent, continuous, real-time speech recognition, targeted at mobile applications. A novel queued memory model, coupled with current and future memory device trends, permits implementation of this architecture using reasonable hardware resources. We have shown such a system can achieve better than real-time recognition rates, making high-performance speech recognition possible for entirely new domains, in which it has the potential to be most useful.

References

[1] IBM ViaVoice, www-306.ibm.com/software/voice/viavoice/

[2] Dragon Naturally Speaking, www.dragonsys.com/naturallyspeaking/

[3] Seymore, K., Chen, S., Doh, S., Eskenazi, M., Gouvêa, E., Raj, B., Ravishankar, M., Rosenfeld, R., Siegler, M., Stern, R. and Thayer, E. The 1997 CMU Sphinx-3 English Broadcast News Transcription System.

[4] Ravishankar, M. K. Efficient Algorithms for Speech Recognition. Ph.D. Thesis, Department of Comp. Sci., Carnegie Mellon University, May 1996.

[5] Rabiner, L. R. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. In Proceedings of the IEEE, Vol. 77, No. 2, February 1989.

[6] Viterbi, A.J. Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm. In IEEE Transactions of Information Theory, Vol. IT-13, pp. 260-269, April 1967.

[7] Hwang, M.-Y. and Huang, X.D. Subphonetic Modeling with Markov States – Senone. In IEEE International Conference on Acoustics, Speech, and Signal Processing, pp. 133-36, March 1992.

[8] Micron Technical Presentation, Memory Subsystem for 2.5G Cellular Handsets, April 2004. www.micron.com/products/presentations.html

[9] Intel White Paper, Key Trends in the Cellular Phone Market for Flash Technology, Spring 2002. www.intel.com/design/flash/papers/