A Low-Power Hardware Search Architecture for Speech Recognition

Patrick J. Bourke, *Rob A. Rutenbar*

Department of ECE, Carnegie Mellon University, Pittsburgh, PA, 15213, USA

pbourke@ece.cmu.edu, rutenbar@ece.cmu.edu

Abstract

High-performance speech recognition is extremely computationally expensive, limiting its use in the mobile domain. We therefore propose a low-power hardware speech recognition architecture for mobile applications, exploiting the orders-of-magnitude efficiency improvements dedicated hardware can offer. Our system is based on the Sphinx 3.0 software recognizer developed at Carnegie Mellon University, capable of large-vocabulary, speaker-independent, continuous, real-time speech recognition. We show through cycle-accurate simulation that our hardware, targeting the backend search stage of recognition, is capable of recognizing speech from a 5,000 word vocabulary 1.3 times faster than real-time, within a 196mW power budget.

Index Terms: hardware, search, speech recognition, low power, circuit

1. Introduction

The very best software speech recognition systems are now approaching their ultimate goal—large-vocabulary, continuous, speaker-independent, real-time speech recognition. While accurate, however, these systems are extremely computationally intensive, requiring the full processing resources of a modern desktop to run in real-time. Such heavy computational requirements either rule out many applications for speech recognition, or require making tradeoffs in accuracy. In particular, low-power or mobile applications are some of those where one might want high-quality speech recognition most—and also those for which orders-of-magnitude performance improvements would be necessary.

One solution to this problem is to migrate speech recognition algorithms from the software-only domain into hardware. Such an approach has proved extremely effective for other tasks, most notably computer graphics. Indeed, application-specific integrated circuits (ASICs) have been shown to allow exactly the orders-of-magnitude performance improvements required in this instance [1].

Recognizing this, there have been several prior efforts to implement speech recognition in hardware. These efforts, however, are either quite dated [2], limited in performance or scope [3,4], or do not consider power consumption [5]. In this paper we therefore propose a low-power hardware search architecture to achieve high-performance speech recognition in silicon. We base our hardware architecture on the successful Sphinx 3.0 software speech recognizer [6].

The organization of this paper is as follows. In the following section we provide a brief background to the Sphinx 3.0 software recognizer, before presenting our hardware architecture in Section 3. In Section 4 we discuss simulation results for this architecture, with particular emphasis on performance and power consumption. Finally, in Section 5 we make some concluding remarks.

2. Background

We choose the Sphinx 3.0 software speech recognizer as a reference for our low-power hardware architecture. Sphinx is a well-known research system developed at Carnegie Mellon University, utilizing continuous Hidden Markov Models (HMMs) to recognize speech, which it processes in 10ms frames. While more recent versions of Sphinx exist, these trade accuracy for performance, making Sphinx 3.0 better suited to our purposes.

The Sphinx recognizer may be conveniently broken down into three processing stages—*feature extraction*, *Gaussian scoring* and *backend search*. In feature extraction, input speech is digitized and undergoes a series of DSP operations to produce a 39-element feature vector for each frame. Gaussian scoring processes this feature vector using a Gaussian Mixture Model (GMM), yielding a corresponding set of senone scores. Finally, in the backend search stage, the Viterbi search algorithm is performed, in consultation with a large trigram language model, to obtain a word hypothesis.

Prior studies [3,5,7] profiling Sphinx have shown that of the three stages identified above, feature extraction processing is negligible, whereas backend search may account for up to 75% of the processing effort in Sphinx 3.0 [5]. Furthermore, each of these studies identify memory access as a major bottleneck in the throughput of speech recognition systems. Accordingly, we concentrate on developing a low-power architecture for the backend search stage in the remainder of this paper, with a particular focus on efficient memory access.

With regard to Sphinx backend search, we note that here too three distinct phases may be identified—*Viterbi scoring*, *transitioning*, and *language model consultation*. Viterbi scoring calculates HMM state probabilities, transitioning combines HMMs to form words, while language model consultation combines words according to the trigram model. We later rely on this division for our hardware architecture.

Two datasets are also of particular importance in backend search. These are the HMMs currently being considered for recognition (which we call the *active HMM list*), and the *word lattice*, which stores potentially recognized words. Together, these two datasets completely specify the state of the Sphinx recognizer at any given time. Indeed, HMM-based speech recognition in Sphinx may be thought of as simply evolving the state of these datasets on a frame-by-frame basis.

Since active HMM list and word lattice data are both read and written, we will refer to these as *dynamic data*—by contrast with the large amounts of read-only model data required for recognition, which we will refer to as *static data*. With these definitions in mind, we now move on to present our hardware architecture.

3. Architecture

In this section we propose a low-power hardware architecture for the computationally complex backend search component of HMM-based speech recognition. We begin first, however, with a brief discussion of our design goals.

3.1. Goals

The primary goal of our hardware architecture is to implement the Sphinx 3.0 backend search, with negligible impact on accuracy, at least real-time performance, and minimal power consumption. To put power consumption in perspective, we consider modern cell phones, a likely platform for speech recognition. Typical power budgets for these devices are around 3W, the majority of which is used for transmission. Sources within the cell phone industry have suggested any feature requiring more than 5-10% of this power budget (150-300mW) would be difficult to justify. Ideally, we would like to see power consumption in the range of 100mW.

Furthermore, there are considerations other than accuracy, speed and power that apply to a mobile system. As both cost and physical size are under great pressure in this domain, any hardware speech recognition system should keep chip area to a minimum. For the same reasons, off-chip resources such as external SRAM, DRAM or Flash will be limited, as will bandwidth and access to those (potentially shared) resources.

Finally, we require that our hardware speech recognition system be scalable. Firstly, a system must be able to scale in terms of the size recognition task it can handle; we consider vocabularies in the order of 5,000-60,000 words. Secondly, a system able to scale well given different power and performance constraints is also desirable.

3.2. Architecture Motivation

As discussed in Section 2, the state of the recognition process in any given frame is completely specified by two datasets—the active HMM list and the word lattice. We note that the active HMM list:

- Is sufficiently large to require off-chip storage.
- Requires the greatest bandwidth of any data accessed during recognition, yet must be read and written with equal measure.
- Contains the data that "drives" the recognition process.

The word lattice, by comparison, is small and infrequently accessed. This suggests a hardware architecture based on the following approach: (1) stream active HMM data on-chip; (2) process each active HMM in turn to yield a new set of active HMMs, in much the same way a conventional processor would process instructions; (3) stream these off-chip again. There are two main advantages to this approach.

First, since active HMM data is read sequentially, it is possible to hide much of the latency in accessing this data, while also taking advantage of burst access modes provided by memories such as DRAM. Second, since it is always possible to look ahead to unprocessed active HMMs, accesses to much of the static speech model data needed to process an active HMM can be effectively scheduled, reducing latency.

It turns out such an approach is feasible, with one small piece of additional machinery. Previously, we identified three main phases in the Sphinx back-end recognition process—Viterbi scoring, transitioning, and language model consultation. While the Viterbi scoring and transitioning phases may be cast in a form suitable for operating on streamed HMMs, the language model phase requires random access to active HMM data. To circumvent this, we introduce a new dynamic memory structure, the *patch list*, which accepts HMM data from the language model [8,9]. This data is then combined with (or *patches*) active HMMs during the scoring phase of the following frame.

3.3. Proposed Architecture

The inputs to our proposed architecture are senone scores generated by Gaussian scoring, and the output a word hypothesis. The architecture is based around three *processing engines*, corresponding to the three Sphinx backend recognition phases. These are the *scoring engine*, the *transition Engine*, and the *language engine* respectively. Figure 1 illustrates our proposed architecture.

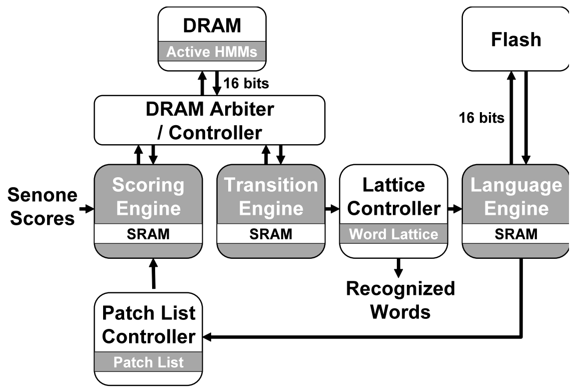


Figure 1: *Proposed low-power architecture.*

Communication between the three processing engines is achieved via the three dynamic memories in the design—the active HMM list, the word lattice, and the patch list. The active HMM list links the scoring and transition engines, the word lattice links the transition engine and language model (while storing likely recognized words), and the patch list links the language model back to the scoring engine. The active HMM list is stored in DRAM, while the word lattice (managed by the lattice controller) and patch list (managed by the patch controller) are small enough to reside in on-chip SRAM.

Recognition of a frame of speech in our hardware architecture proceeds as follows. The scoring engine first reads in active HMMs from the previous frame, calculates new state probabilities for each using the current senone scores, and writes these back to DRAM. Each newly scored HMM is then read by the transition engine, where it is either pruned, written back to DRAM alone, or written back to DRAM with any additional HMMs to which it may transition. While processing each active HMM, the transition engine notes any that complete a word, and generates a word lattice entry if necessary. These word lattice entries are then used by the language engine to determine which words should continue to be considered. Data for such words are entered into the patch list, and used to update the active HMM list during scoring of the next frame.

Thus far, we have only discussed dynamic memory in our architecture. There remains the large quantity of static speech model data required by each processing engine to perform its task. As shown in Figure 1, this is stored in either on-chip SRAM, or external flash memory for the trigram language model (due to its size). For a standard 5,000 word benchmark [10], on-chip SRAM requirements are 11.7Mbit, divided amongst 46 individual SRAMs—certainly a large amount, but not without precedent in today's technologies. The six largest on-chip SRAMs account for 8.8Mbit of this total.

For the same 5,000 word benchmark, the trigram language model requires 21MByte of external flash memory. In order to efficiently locate trigrams within this memory, we employ a novel cuckoo hashing technique [11]. Two hash functions are used, with two bins per hash. We choose CRC codes [12] as hash functions since these may be calculated extremely efficiently in hardware.

4. Results and Analysis

We begin this section by briefly discussing our design methodology and some implementation details for our architecture. We then evaluate our design in terms of accuracy, performance, and power consumption.

4.1. Design Methodology

In implementing our architecture we have several, somewhat conflicting goals. Firstly, we are concerned with accuracy, and in particular an ability to accurately determine recognition speed and power consumption. Accordingly, we decided on a register-transfer level (RTL) implementation. Secondly, we desire the ability to modify our design with relative ease in order to explore different architectural tradeoffs. Thirdly, we must be able to simulate large datasets. Finally, we would like a design methodology that offers a clear path to creating (eventually) a real logic-level netlist. Weighing these, we chose to implement our architecture using SystemC, since:

- SystemC provides many features useful in modeling our design with a high degree of accuracy, e.g., clocks, register-like data structures.
- As SystemC is simply a C++ library, existing C++ simulator code could be made use of.
- Simulation is relatively fast (compared to, say, Verilog).

While choosing to work at the register-transfer level does have an impact on the flexibility of our simulator, this provides significant advantages in terms of accuracy, and in later moving to an actual hardware implementation (i.e. Verilog). Unfortunately, logic synthesis directly from SystemC is not a particularly mature process at this time.

In the remainder of this section, we present results based on simulation of speech selected from the 5,000 word *Wall Street Journal* (WSJ) corpus [10], of which we simulate approximately one hour, or a little under 400,000 frames. We assume a modest clock speed of 100MHz for our simulations.

Selected implementation details for this design configuration are shown in Table 1.

Table 1. *Selected implementation details.*

	Scoring Engine	Transition Engine	Language Engine
State Machines	5	5	6
Pipelines	1 x 4 stage 1 x 5 stage	1 x 2 stage	None
Register Bits	4,230	1,384	2,407
SRAMs	15	8	20
SRAM Size	7.1 Mbit	960 Kbit	3 Mbit
Code Size	3,438 lines	1,460 lines	7,534 lines

4.2. Evaluation: Accuracy

Our design currently yields results identical to the Sphinx 3.0 software recognizer. We verify this by comparing the contents of the two datasets that describe the state of each recognizer—the active HMM list and the word lattice. For the 5,000 word WSJ task we therefore maintain the same word error rate as a similarly configured Sphinx 3.0, or 6.707%.

4.3. Evaluation: Performance

At our chosen clock speed of 100MHz, we assume at most a single on-chip SRAM read and integer arithmetic operation may be performed per cycle. We assume 16-bit wide interfaces to both DRAM and flash, and model these memories after current commercial parts [13,14]. DRAM is simulated using DRAMsim [15].

Across the one hour of speech simulated, we achieve a recognition rate of 0.77x real-time (RT), 1.3 times faster than real-time. Alternatively, we run 1.4 times faster than Sphinx 3.0, with a clock speed 28 times slower, and using vastly fewer resources. Access to the active HMM list stored in DRAM limits our performance quite severely; if active HMMs could be read in a single cycle, our recognition rate would increase approximately six-fold.

During recognition, the active HMM list fluctuates in size depending on the current complexity of input speech. Typically the active list occupies around 0.4MB; 1MB of external DRAM would be sufficient to store 99.99% of all active lists encountered.

4.4. Evaluation: Power

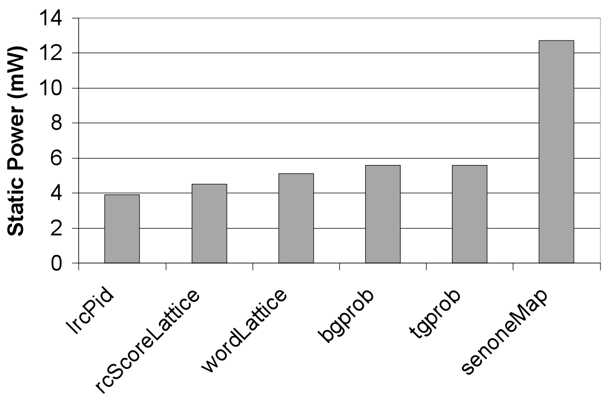
We estimate the power consumption of our backend search architecture by breaking this into its constituent components and considering each separately. These components are power due to: arithmetic operations; SRAM accesses; DRAM accesses; flash accesses; memory controller operations; control logic; and clock, register and pin power. We discuss each in turn.

The power consumption of individual arithmetic operations is determined by scaling values measured for similar operations in 90nm CMOS technology [16]; we then simply count operations. The arithmetic involved is not particularly complicated—the most complex operation is 6-bit constant multiplication—and comes to only 0.43mW total. This is in agreement with our earlier assertion that it is memory operations that are most important in any hardware speech recognition system.

In stark contrast, the large amount of on-chip SRAM in our design is responsible for the majority of power consumed. We consider the dynamic and static power contributions of SRAM separately. Dynamic power (i.e. power due to reads and writes) is based on SRAM compiler output for a 130nm CMOS node [17] to which we have access, scaled appropriately for 90nm, and comes to only 0.56mW. Using Cacti 5.0 [18], however, we find that the static power consumed by 11.7Mbit of on-chip SRAM comes to 71mW. The six largest static power contributions by individual SRAMs are shown in Figure 2.

External DRAM power is determined in the manner of DRAMsim [15] and totals 70mW for our chosen device [13]. Flash memory power is determined by considering the proportion of clock cycles in which the memory is active, while also taking into account burst and sleep modes, and totals 11.3mW. We take DRAM controller power to be 11mW [19].

Figure 2: *Static SRAM power consumption.*



We assume 10,000 gates of control logic, consuming 10mW. Clock and register power consumption is determined using the method outlined in [16]. Performing this calculation for a 100MHz clock and 4,700 total register bits yields a result of 12.5mW. We calculate pin power at 9mW [19]. The contributions of each power component to the total power consumption of our design are shown in Figure 3.

4.5. Analysis

In agreement with initial expectations, our results show that access to memory is the predominant factor in determining both the performance and power consumption of our design. In particular, arithmetic operations (i.e. actual computation) contribute little to total power. For the 5,000 word WSJ task, our proposed design achieved a recognition rate of 0.77x RT—40% faster than Sphinx 3.0 on our reference system—while using vastly fewer resources and reducing power consumption by roughly two orders of magnitude.

At present, our design requires 11.7Mbit of on-chip SRAM and consumes an estimated 196mW. While this design is certainly not unrealistic given current manufacturing technologies, it would be far more attractive with significantly less on-chip SRAM, and power consumption closer to 100mW (recalling again that a full recognizer will require power for feature extraction and Gaussian scoring). Of course, these problems are related, as static SRAM power makes the largest contribution to total power consumption, which suggests minimizing on-chip SRAM use to be an important future research goal.

5. Conclusion

In this work, we proposed implementing the backend search for high-performance speech recognition in silicon, in particular with mobile applications in mind. We presented a first-generation low-power hardware architecture for doing so, and simulation results showing such an architecture can meet the stringent demands of mobile systems where software could never hope to do so. While much research remains in this area, we believe a migration to silicon offers the possibility of many novel applications for speech recognition technology.

6. Acknowledgements

This research was supported by the NSF and the Center for Circuit & System Solutions (C2S2), one of five centers in the Focus Center Research Program, an SRC program.

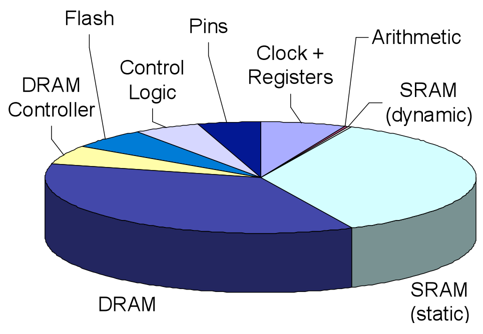


Figure 3: *Power consumption breakdown.* ***196mW*** *total.*

7. References

[1] Brodersen, R., "Low-Voltage Design for Portable Systems", IEEE Solid State Circuits Conf., Feb. 2002.

[2] Stölzle, A., Narayanaswamy, S., Murveit, H, Rabaey, J.M. and Brodersen, R.W., "Integrated Circuits for a Real-Time Large-Vocabulary Continuous Speech Recognition System", IEEE J. Solid-State Circuits, 26(1):2-11, Jan. 1991.

[3] Mathew, B., Davis, A. and Fang, Z., "A Low-Power Accelerator for the SPHINX 3 Speech Recognition System", Intl. Conf. on Compilers, Architecture and Synthesis for Embedded Systems, 2003.

[4] Krishna, R., Mahlke, S. and Austin, T., "Architectural Optimizations for Low-Power, Real-Time Speech Recognition", Intl. Conf. on Compilers, Architecture and Synthesis for Embedded Systems, 2003.

[5] Lin, E.C., Yu, K., Rutenbar, R.A., and Chen, T., "Moving Speech Recognition from Software to Silicon: the In Silico Vox Project", Proc. Interspeech 2006, Sep. 2006.

[6] Ravishankar, M. K., "Efficient Algorithms for Speech Recognition", PhD Thesis, Dept. of Computer Science, Carnegie Mellon Univ., May 1996.

[7] Agaram, K., Keckler, S.W. and Burger, D., "A Characterization of Speech Recognition on Modern Computer Systems", Proc. 4th Workshop on Workload Characterization, Dec. 2001.

[8] Bourke, P.J., "A Queue-Based Architecture for Hardware Speech Recognition", M.S. Thesis, Dept. of ECE, Carnegie Mellon Univ., Aug. 2004.

[9] Bourke, P.J. and Rutenbar, R.A, "A High-Performance Hardware Speech Recognition System for Mobile Applications", Proc. SRC Techcon, 2005.

[10] Pallett, D.S., "A Look at NIST's Benchmark ASR Tests: Past, Present, and Future", Proc. 2003 IEEE Workshop on Automatic Speech Recognition and Understanding, 2003.

[11] Pagh, R. and Rodler, F.F., "Cuckoo Hashing", 9th Annual European Symposium on Algorithms, v.2161 Lecture Notes in Computer Science, pp. 121-133, Springer-Verlag, 2001.

[12] Peterson, W. W. and Brown, D. T., "Cyclic Codes for Error Detection", Proc. IRE, Jan. 1961.

[13] Samsung Electronics, Inc., "K4X28163PH: 8M x 16 Mobile-DDR SDRAM", Oct. 2005.

[14] Samsung Electronics, Inc., "K8C5715ETM: 16M x 16 Synchronous Burst, Multi Bank NOR Flash Memory", Oct. 2006.

[15] Wang, D., Ganesh, B., Tuaycharoen, N, Baynes, K., Jaleel, A. and Jacob, B., "DRAMsim: A Memory-System Simulator", SIGARCH Computer Architecture News, 33(4):100-107, Sep. 2005.

[16] Markovic, D.M., "A Power / Area Optimal Approach to VLSI Signal Processing", PhD Thesis, Dept. of EECS, Univ. of California at Berkeley, May 2006.

[17] TSMC, "0.13µm High-Speed Single-Port Synchronous SRAM", 1991.

[18] Tarjan, D., Thoziyoor, S. and Jouppi, N.P., "Cacti 4.0", Tech. Report HPL-2007-167, Hewlett-Packard Laboratories, Palo Alto, Oct. 19, 2007.

[19] Horowitz, M., Personal communication, Stanford Univ., Sep.-Oct. 2007.