

## A Large-Vocabulary Real-Time Continuous-Speech Recognition System

H. Murveit, J. Mankoski,  
SRI International, Menlo Park, CA

J. Rabaey, R. Brodersen,  
T. Stoelzle, D. Chen, S. Narayanaswamy, R. Yu, P. Schrupp,  
University of California, Berkeley, CA

R. Schwartz  
Bolt, Beranek, and Newman, Cambridge, Mass.

A. Santos  
ETSI Telecommunicacion, 28040 Madrid, Spain

### ABSTRACT

This paper describes an approach to implementing speech recognition systems with very high throughput rates by using custom designed datapaths for time-critical computations. Using this approach, we are currently designing and debugging a real-time 3000-word continuous-speech recognition system that uses bigram language models. Logical extensions to this prototype should be able to implement systems with an order-of-magnitude more words.

### INTRODUCTION

In the last few years, hidden-Markov-model-based (HMM) algorithms [Wei89], [Cho87],[Jel85], [Lee88],[Pau87],[Rab86] have been the most successful techniques used for speech recognition systems. We have developed a system architecture to implement real-time large-vocabulary continuous-speech recognition using HMM algorithms and bigram language models. This architecture will run (in real time) a class of such HMM-based systems that should include expanded versions of the DECPHER system [Wei89] being developed at SRI International, the BYBLOS system [Cho87] being developed at Bolt, Beranek, and Newman, and the SPHINX system [Lee88] being developed at Carnegie-Melon University. This architecture can implement speech recognition systems with the following specifications:

- Continuous-speech input
- Large vocabulary recognition
- Viterbi search HMM speech recognition algorithm
- Up to four independent discrete output distributions per state
- Bigram language modeling.

The overall system architecture is shown in Figure 1. The major processing is achieved with three separate modules and is controlled by a microcomputer:

- The front-end module samples the speech and extracts basic features.

- The word-processor module performs a Viterbi search to find the probabilities that words in its vocabulary match features corresponding to portions of the input speech. The probability that vocabulary words end at each time interval is sent to the grammar processor.
- The grammar processor transmits the probability that each vocabulary entry might start at each current time interval to the word processor (based on probabilities that previous words had finished at the previous time interval, and probabilities of word-to-word transitions).

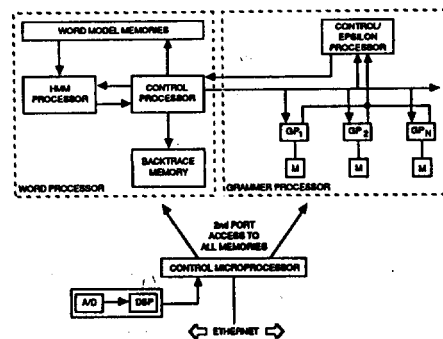


Figure 1. System Architecture

### ALGORITHMS

We refer the reader to several papers in the references which describe HMM-based speech recognition algorithms in detail (for example [Jel85] or [Rab88]). In this paper we focus on two of the computationally most demanding portions of the HMM algorithms.

#### The HMM Search

One of the central parts of the HMM algorithm involves the computation of state probabilities  $P(s,t)$ .

The system uses these probabilities to find the best sequence of states (and hence the best sequence of words) that corresponds to spoken input. The following recursion can be used to compute these probabilities:

$$P(s,t) = \underset{\hat{s}}{\text{MAX}} \left[ P(\hat{s},t-1) \cdot A(\hat{s},s) \right] B(s,O_t) \quad (1)$$

where  $s$  is a state in an HMM network,  $\hat{s}$  as a state in the network that can precede  $s$ ,  $A(\hat{s},s)$  is the Markov state transition probability from state  $\hat{s}$  to state  $s$ , and  $B(s,O_t)$  is the probability that state  $s$  would generate the speech features measured by the recognition system at time  $t$ .

In speech recognition systems such as DECYPHER or BYBLOS, an average word might have 18 states, and speech features might be measured every 10 milliseconds. Therefore, if Equation (1) were processed every 200 nanoseconds,<sup>1</sup> and there were no other system bottlenecks,  $10^{-2}/(18 \times 2 \times 10^{-6}) = 2777$  whole word hypotheses could be updated in real time.<sup>2</sup>

However, if Equation (1) were processed in 200 nanoseconds, all the data required to solve that equation would have to be read from memory in that same time interval. Using numbers based on bit widths in our implementation, this would be 16 bits for each  $P(\hat{s},t-1)$ , 8 bits for each  $A(\hat{s},s)$ , 8 bits for  $B(s,O_t)$ , and 16 bits to write back  $P(s,t)$ . Assuming that states have an average of three predecessors, this would come out to 96 data bits. Additional data bits would also be required to read and store tag values (18 bits for each predecessor) that allow the reconstruction of the recognized word sequence. Finally, there would be a similar number of address bits required for a system with a large state space. Therefore, several hundreds of bits of memory address and data pins are required for straightforward implementations, rendering this a difficult task for standard microprocessor technology.

### The Bigram Search

Although all state-to-state transitions in an HMM-based speech recognition system can be processed as described above, we prefer to separately handle those transitions that go from states at the ends of words to states at the starts of words. We do this in order to maintain an assumption we require below in our description of the word processor, namely, that predecessors of a state must be *near* the state. In general, this assumption is not true for across-word transitions.

Assuming that words only had one initial state, then one algorithm to compute the best across-word predecessor probability for such states would be to compute

<sup>1</sup>Our initial designs are, for simplicity, based on machine cycle times that match standard dynamic RAM cycle times, about 200 nanoseconds. Future designs may use interleaving or other techniques to improve systems throughput.

<sup>2</sup>For systems with no pruning of hypotheses (such as our first prototype), this would be the allowable real-time vocabulary size. With hypothesis pruning, this may be up to an order of magnitude less than the vocabulary size.

$$P_{start}(w_i,t+1) = \underset{\hat{w}}{\text{MAX}} \left[ P_{start}(w_i,t+1), P(\hat{s},t) \cdot A(\hat{s},out) \cdot P(w_i|\hat{w}) \right] \quad (2)$$

where  $P_{start}(w_i,t+1)$  is initially zero for all  $i$  at the start of time slot  $t$ . Here,  $\hat{s}$  is a final state in word  $\hat{w}$ ,  $P(\hat{s},t)$  is that state's probability,  $A(\hat{s},out)$  is the probability that that state will transition outside  $\hat{w}$ , and  $P(w_i|\hat{w})$  is the probability that word  $w_i$  will succeed word  $\hat{w}$ . Then  $P_{start}(w_i,t+1)$  in Equation (2) could be treated as  $P(\hat{s},t-1)$  for the evaluation of Equation (1) during the next time slot for the virtual state  $\hat{s}$  representing the best across-word predecessor of an initial state  $s$  in word  $w_i$ .

If Equation (2) could be processed every 200 nanoseconds, then 50,000 across word transitions could be updated every 10 millisecond interval.

With a 3000-word vocabulary, and all 3000<sup>2</sup> bigram probabilities allowed, then 9,000,000 across-word transitions would need to be updated every 10 ms interval. However, many of the transitions are low-probability and can be approximated with a simpler scheme[Rab88]. Furthermore, a pruning scheme can be implemented to avoid processing across-word transitions whose resultant  $P_{start}(w_i,t+1)$  would be lower than a threshold. We expect that hardware than can process up to 200,000 across-word transitions per frame will be adequate.

### SPECIAL PURPOSE DATHPATHS

We have designed new integrated circuits with special-purpose datapaths and multiple memory ports to implement the above equations.

### HMM Search

Figure 2 shows a datapath implementing Equation (1). This datapath has several interesting features.

**LOG COMPUTATIONS:** Since the operations are multiplications and comparisons, the arithmetic is done in the log domain and adders can then be used instead of multipliers.

**PARALLELISM:** We take advantage of the fact that in many HMM-based speech recognition systems, most states have three or fewer predecessors. We provide complete datapaths to perform the calculations inside MAX term of Equation 1 for each of three possible predecessors per state in parallel. If a state has more than three predecessors, then it will take more than one cycle to compute Equation 1.

**PIPELINING:** The datapath is heavily pipelined (13 stages) so that the computational units can process their data in one machine cycle. Due to the nature of the computations and the lack of conditional branching at a low level, the pipeline only needs to be emptied and refilled at the start of a 10 millisecond frame, therefore there is virtually no overhead associated with filling and emptying the pipeline.

**REDUCED MEMORY ACCESS:** The architecture takes advantage of the fact that all within-word transitions are local, and that one can order the states within a word so that a higher numbered state will never make a within-word transition to a lower numbered state.<sup>3</sup> The circuit in Figure 2 reads the Probabilities  $P(\hat{s}, t-1)$  from the stateprob(t-1) memory and loads the value concurrently into separate register banks that are configured as shift registers. A topology memory provides the identity of the predecessors  $\hat{s}$  of state  $s$  and the transition probabilities  $P(s|\hat{s})$ . The predecessor identities are used as addresses into the three parallel register banks so that the terms  $P(s, t) \cdot A(s|\hat{s})$  can be computed and compared for three predecessors  $\hat{s}$  in parallel. Computation proceeds for word initial to word final states using the partial ordering of states described above. Because of the ordering, it can be guaranteed that the proper  $P(\hat{s}, t-1)$  values will be in the register banks when needed for the computation of  $P(s, t)$ , as long as the predecessor state  $\hat{s}$  is numbered within  $N$  states of  $s$  where  $N$  is the size of the register bank. Thus, the use of this scheme reduces the number of memory access into the stateprob memory by a factor of up to three.

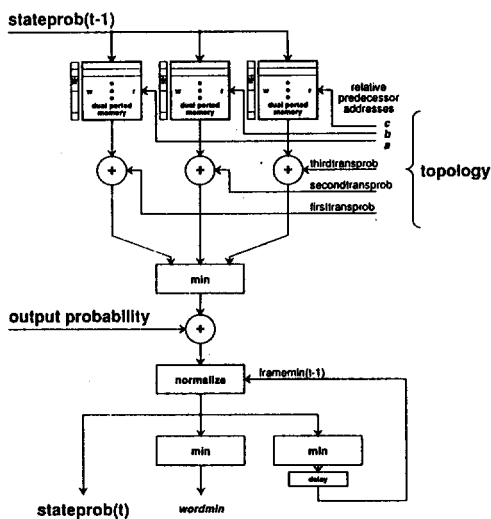


Figure 2. Viterbi Search Datapath

### Bigram Search

The top level description of a system which implements across-word transitions, incorporating bigram grammars, is shown in Figure 2. Here there are four processors with separate memories, and another control processor. This discussion will focus on the four parallel processors each of which can compute Equation (2) in one machine cycle (200 ns), and thus a throughput of 200,000 across-word transitions is possible.

<sup>3</sup>This assumption, that there are no within-word cycles containing more than one state, is true for all HMM based speech recognition systems that we are aware of.

The processors are implemented with a simple finite-state machine controlling a special-purpose data path. The machine is input the probability that word  $w$  completed at time  $t$ . Given the new information, its objective is to update the probability  $P_{start}(w_i, t+1)$ , (Equation 2), for all words  $w_i$  that can succeed  $w$ . When implementing Equation (2) for one of the  $w_i$  the processors perform the following tasks in one machine cycle:

- (1) reads a value for  $P_{start}(w_i, t+1)$ , where  $w_i$  is the  $i^{th}$  most likely word to follow  $\hat{w}$ .
- (2) adds the log of  $P(\hat{s}, t) \cdot A(\hat{s}, OUT)$ , which is stored in an internal register at the start of a processing for the final-state  $\hat{s}$ , to the log of  $P(w_i|\hat{w})$ ,
- (3) compares the computed and looked-up values and store the greater on in  $P_{start}(w_i, t+1)$ ,
- (4) compares  $P_{start}(w_i, t+1)$  with a threshold to decide if more of the successors of  $\hat{w}$  should be processed.

The grammar processor has many features in common with the word processor. It relies heavily on pipelining (9 stages) to achieve high speed computation. It implements its multiplies as log adds. FIFO buffering is used extensively in communications between processors to maintain asynchronous computation. However, there are several features about the bigram processor's implementation that distinguish it from that of the word processor.

**SINGLE-CYCLE READ/WRITE:** Because Steps (1) and (3) read and write to the same memory location ( $P_{start}(w_i, t+1)$ ), high speed static RAMS must be used to allow two memory cycles in a single machine cycle. Fortunately,  $P_{start}(w_i, t+1)$  is a small memory. It only needs a single  $w_i$  entry for each word in the vocabulary, and only the times  $t$  and  $t+1$ .

**PARALLELISM WITHOUT REPLICATION OF MEMORY:** Although parallel processors are used that all access the same logical memories ( $P_{start}(w, t+1)$  and  $P(w_i|\hat{w})$ ) at the same time, these memories need not be replicated. Rather, the memory needs to be partitioned into separate physical memories for each processor, but each memory needs only to be  $1/N$  times the size of the overall memory for  $N$  processors. This is important because  $P(w_i|\hat{w})$  may be one of the largest memories in the system.

### SYSTEM IMPLEMENTATION

To implement the architecture described in this paper, five custom integrated circuits (ICs) were designed using the LAGER IC design tools developed at U.C. Berkeley, and manufactured by DARPA's and NSF's MOSIS facility.

Three ICs are part of the word-processor, and two are part of the grammar processor. The largest IC is the Viterbi-search IC (whose main datapath is shown in Figure 2). It has 204 pads, is  $10 \times 11.5$  millimeters, has 25,000 transistors, and has 13 levels of pipelining. Two other ICs designed for the word-processor each have 134 pads and are  $6.8 \times 7.5$  mm. and  $6.8 \times 6.8$  mm.

One IC in the grammar processor computes Equation (2). It has 204 pads, 11,300 transistors, and is 9.6×10 mm. The other computes deals with low probability across-word transitions and sends  $P_{start}$  back to the word processor. It has 156 pads, 9500 transistors, and is 9.8×9.6 mm.

All ICs have been designed and manufactured and are currently being tested.

Two printed circuit boards are currently being designed to implement the word and grammar processors. The system and ICs were simulated using the THOR and IRSIM simulation programs.

#### Future Plans

The initial design currently being designed and debugged will recognize approximately 3,000 words in real time and can process 200,000 across-word transitions when using four bigram processors. We have been somewhat conservative in our initial implementation in order to improve the chances for early-working hardware. Two fairly straight-forward modifications can increase the processing power dramatically. Cycle times of 100 nanoseconds are possible with interleaved memory configurations, because the memory access of dynamic RAMS in the system is typically sequential. The across-word and within-word processors can be modified to process only promising word hypotheses and prune the more unlikely words from consideration. The combination of these two techniques in a future prototype would likely result in a vocabulary size of about 20,000 words.

#### CONCLUSIONS

A large-vocabulary real-time continuous-speech recognition system has been described. It has been shown that the largest bottleneck in such a system is located in the access of the memories. The architecture exploits a variety of techniques, such as partitioning and replication in order to cope with this memory bottleneck. The required throughput is achieved with the aid of extensive pipelining (up to thirteen levels deep) and concurrency. The architecture allows for the extension to larger vocabularies by just adding more parallel units. Pin count considerations have resulted in the definition of five custom integrated circuits, which are currently being tested.

#### REFERENCES

- [Cho87] Y.L. Chow, M.D. Dunham, O.A. Kimball, M.A. Krasner, G.F. Kubala, J. Makhoul, P.J. Price, S. Roucos, and R.M. Schwartz, "BYBLOS: The BBN Continuous Speech Recognition System," *Proc. ICASSP-87* pp. 89-92 April 1987
- [Jel85] F. Jelinek, "The Development of an Experimental Discrete Dictation Recognizer", *IEEE Proceedings*, Vol. 73, No 11, pp. 1616-1624, Nov. 1985.
- [Lee88] K. Lee and H. Hsiao-Wuen, "Large Vocabulary Speaker-Independent Continuous-Speech Recognition Using HMM," *Proc. ICASSP 88* pp. 123-126 April 1988
- [Pau87] D.B. Paul, "A Speaker-Stress Resistant Isolated-Speech Recognizer," *Proc. ICASSP 87* pp. 713-716 April 1987
- [Rab86] L. Rabiner and B. Huang, "An Introduction to Hidden Markov Models", *IEEE ASSP Magazine*, pp. 4-16, Jan. 1986.
- [Rab88] Rabaey J., Brodersen, R., Stoelzle, A., Chen, D., Narayanaswamy, S., Yu R., Schrupp, P., Murveit, H., and A. Santos, "A Large-Vocabulary Real-Time Continuous-Speech Recognition System," in *VLSI Signal Processing, III*, edited by R.W. Brodersen and H.S. Moscovitz, IEEE Press, New York, NY, 1988
- [Wei89] Weintraub, M., Murveit, H., Cohen M., Price, P., Bernstein, J., Baldwin, G., and D. Bell, "Linguistic Constraints in Hidden Markov Model Based Speech Recognition," *Proc. ICASSP 89 Jersey*,