

A VLSI Wordprocessing Subsystem for a Real Time Large Vocabulary Continuous Speech Recognition System

A. Stölzle, S. Narayanaswamy, K. Kornegay, J. Rabaey, R. W. Brodersen
University of California at Berkeley, Dept. of EECS

Abstract

This paper presents the architecture of a word processing subsystem for a large vocabulary real time continuous speech recognition system. The system contains three application specific integrated circuits to perform the Viterbi algorithm for 50,000 states using a breadth first recognition search in real time. This corresponds to a computation rate of 225 Megaoperations per second excluding memory operations where 670 Megabits have to be accessed per second.

1 INTRODUCTION

It is generally accepted that Hidden Markov Models (HMMs) are currently the most efficient technique to model speech for use in automatic speech recognition [8,5,1,6,2]. It has largely replaced the technique of modelling speech with a concatenation of linear time invariant systems (LTI systems) along with pattern matching using the dynamic time warp algorithm [4,3]. The raw concept of modeling speech using Hidden Markov Models is to exploit the fact that speech has periods of rather steady behavior. The temporal variation of the speech signal can be modeled by concatenating these periods of steady behavior (states) using transitions with an associated transition probability.

The speech recognition system this paper is concerned with [7] implements in real time a class of HMM-based speech recognition systems based on the BYBLOS system developed by Bolt Beranek and Newman [2].

The system is partitioned in two subsystems exploiting the hierarchical nature of the HMM: the word processing subsystem that works on the HMM of individual words and the grammar processing subsystem that works on a composite HMM describing the concatenation of all the words in the recognition vocabulary.

This paper describes the algorithm, architecture and implementation of the word processing subsystem. This system

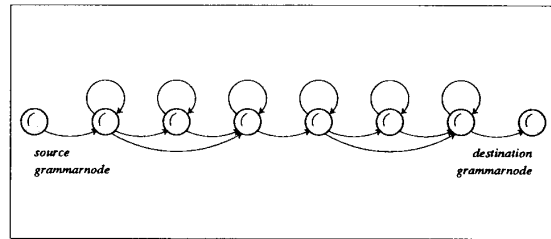


Figure 1: State transition graph of a typical HMM used in the word processing subsystem

meets the ambitious computational requirements by using special purpose processors with dedicated pipelined datapaths and parallelism to implement the most time critical parts of the recognition algorithm. To deal with the vast amount of data that has to be accessed an architecture featuring parallel on chip bidirectional memories is used.

2 ALGORITHM

A Hidden Markov Model is a process that is composed of two elements. The first element is a Hidden Markov Chain that can be described with a directed graph (see Fig.1). The nodes in this graph reflect the states $s_1 \dots s_N$ of the HMM and associated with the arcs (transitions) are transition probabilities, $a_{i,j}$ for the transition of state s_i to state s_j .

The second element is a set of probability distributions, where each distribution is associated with a particular state in the Hidden Markov Chain.

For each (discrete) instance of time these probability distributions generate *output probabilities* $b_{j,k}$ based on the incoming signal (speech). A particular $b_{j,k}(t)$ is the probability that state s_j matches the input symbol $v_k(t)$. In our system, $v_k(t)$ is a vector quantized feature of the incoming speech $\alpha(t)$ for a particular timeframe t .

The Markov Models used in this system are of the order one. That means, a transition from one state to another only

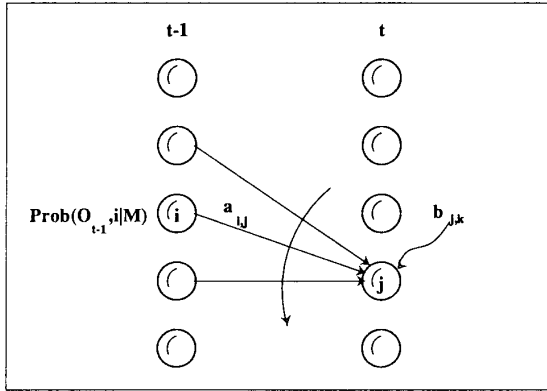


Figure 2: Lattice structure implementing the Viterbi algorithm on the word level

depends on this state and the predecessor state. Also, the observations are drawn from a finite set of symbols which corresponds to discrete probability distribution functions associated with each state.

The word processing subsystem uses left-to-right HMMs. This feature of Markov Models means, that there is no transition to any given state from a state situated to the right. In other words, all the *predecessors* of a state are to the left to this state (see Fig. 1).

In summary, a HMM can be described with an initial state distribution vector $\pi' = \pi_1, \pi_2, \dots, \pi_N$ that gives the probabilities that the process is in a particular state at time $t=0$, a set of transition probabilities $a_{i,j}$ and a set of discrete probability distributions $b_{j,k}$.

2.1 Viterbi Algorithm

The Viterbi algorithm is used to find the single best state sequence through a HMM given an input sequence $O_T = o(1)..o(T)$. This state sequence $S_{s_j(T)} = s(1)..s(T)$ is the path that maximizes the probability $Prob(O_T, s_j | M)$, the single best state sequence through the Hidden Markov Model M that ends in state s_j given O_T . Since there is no need to quantitatively compute the probability of the best path, there is an efficient way to find this state sequence using dynamic programming.

Let us define $Prob(O_t, s_j | M)$ as the probability of the most probable state sequence given the t inputs $O_t = o(1), o(2)..o(t)$ that ends with the state s_j . Furthermore, let $S_{s_j(t)} = s(1), s(2)..s(t)$ be the corresponding state sequence.

Then we can start the recursion

$$Prob(O_t, s_j | M) = b_{j,k}(t) \cdot \max_i [Prob(O_{t-1}, s_i | M) \cdot a_{i,j}] \quad (1)$$

with the initial value

$$Prob(O_1, s_j | M) = \pi_j b_{j,k}(1) \quad (2)$$

To store the state sequence we use equation (3):

$$S_{s_j(t)} = \{S_{s_p(t-1)}, s_j\} \quad (3)$$

This equation describes a concatenation of the state sequence $S_{s_p(t-1)}$ with s_j where s_p is the state that maximizes equation (1) and $S_{s_p(t-1)}$ is the state sequence that ends in state s_p .

The implementation of these equations for HMMs modelling words leads to a lattice structure as denoted in Fig.2. $Prob(O_t, s_j | M)$ can be computed for all the states and for all the observations $o(t)$ so that after the last observation $Prob(O_T, s_j | M)$ for all states $s_j \in \{s_1..s_N\}$ is readily available. The termination of this recursion then is to find the state s that maximizes

$$\max_j Prob(O_T, s_j | M) \quad (4)$$

This produces the most probable sequence of states $S_s(T)$ with the probability $Prob(O_T, s | M)$. This is the solution to the task of recognizing speech, namely to find the most probable state sequence through a composite HMM given a sequence of observations $O_T = o(1), o(2)..o(T)$

3 SUBSYSTEM ARCHITECTURE

Fig.3 shows a high level block diagram of the word processing subsystem. It implements the Viterbi algorithm by working on the lattice structure shown in Fig.2. For every time frame t the probabilities $Prob(O_t, s_j | M)$ of all the states s_j in the system are being updated using eq.(1).

Due to the large amount of memory required in this system (≈ 12 Mbyte) all the memories are implemented using dynamic RAMs. They are partitioned in the following way (see Fig.3):

The topology memory stores the transition probabilities $a_{i,j}$ between all the states in the recognition vocabulary.

The output memory and the output memory lookup table are an implementation of the discrete probability distributions associated with each state. The function of the output memory lookup table is to minimize the storage requirement of the output memory. Since not every state in all the HMMs describing the vocabulary words are unique states (a particular random function might be valid for several states in the models), only the $b_{j,k}$'s of unique states j are stored in the output memory. The lookup table then gives the mapping between the states in the model and the unique states. The output memory can be configured in such a way that up to 4 different features of a particular speech frame can be used to derive an output probability (multiple outputs).

The state probability memories t and $t-1$ store the probabilities of all the states at times t and $t-1$, $Prob(O_t, s_j | M)$ and $Prob(O_{t-1}, s_i | M)$ (see equ.(1)). Basically, $Prob(O_{t-1}, s_i | M)$ is being read from one memory

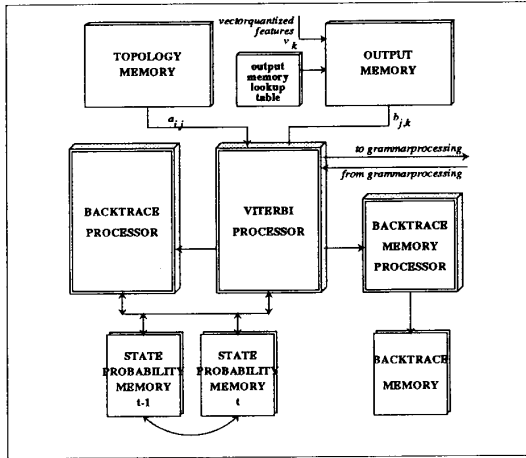


Figure 3: Block diagram of the word processing subsystem.

while the new probabilities $Prob(O_t, s_j|M)$, computed using eq.(1), are being written to the other memory. After this has been done for all the states, the memories get swapped in such a way that the memory containing $Prob(O_t, s_j|M)$ is being read and the other memory written.

The backtrace memory stores all the N state sequences $S_{s(t)}$ terminating in the N states $s_1 \dots s_N$. The sequences are coded using a single linked list where each element contains a pointer (memory address) to the next element stored at the previous frame along with an identification of the state. After the termination of the Viterbi algorithm the single best path can be recovered by traversing the linked list (reading the backtrace memory) starting from the state that maximizes $\max_j Prob(O_T, s_j|M)$.

Since all the memories throughout the word processing subsystem are DRAMs, it is desirable to access all the data needed to perform (1) in a sequential fashion. This makes it possible to have fast access of data by using the fast page access mode.

Since updating the probabilities of the states at time t is done sequentially for all states in the models, the following memories can be accessed sequentially: The state probability memory storing $Prob(O_t, s_j|M)$ and the topology memory that stores all the information about the location of the predecessor states along with their transition probabilities. However, the memory that stores the probabilities $Prob(O_{t-1}, s_i|M)$ has to be randomly accessed for read. Also, due to the output memory lookup table the output memory storing all the $b_{j,k}$ is accessed randomly.

To facilitate accessing the state probability memory $t-1$ that stores $Prob(O_{t-1}, s_i|M)$, we exploit the fact that the HMMs used throughout this subsystem are left to right HMMs. That means, if the probability of a given state has to be updated,

no state in the graph left to this state has to be accessed or, no memory location with an address that is bigger than the address of the current state has to be accessed. Also, since the transitions between states are relatively local transitions (there is no predecessor state that is further than 15 states from the current state) there is only a small subset of the state probability memory $t-1$ that is accessed randomly. Thus, if an on-chip memory in the Viterbi Processor that stores the 16 $t-1$ probabilities of the states preceding the current state is used, it is possible to have a fast random access on the chip while the transfer of data between the external memory and the Viterbi Processor is done sequentially. This method also makes it possible to parallelize computation on a single chip without increasing the bandwidth: It is only necessary to have multiple copies of the on chip memories having the same content so that there is no read contention if several data have to be read simultaneously.

A possible solution to eliminate the random access of the output memory is to store all the $b_{j,k}$ for all states in the vocabulary. However, the tradeoff is that memory would be wasted storing duplicate information. We decided to use the lookup table approach and to only store the probabilities $b_{j,k}$ associated with unique states. This implementation requires 8 Mbyte of memory while the other approach that allows sequential access would require 64 Mbyte. Accessing this output probability memory thus sets the system speed (5MHz).

The task of creating a linked list that describes the state sequences so that the most probable sequence can be recovered is implemented in the following way: Every probability of a state has a tag associated with it pointing to the predecessor state that maximizes eq.(1). Physically, these tags are stored in the same memory location where the state probabilities are stored. So whenever a probability of a state is being updated, the associated tag has to be updated. This operation is performed in the Backtrace Processor: This processor reads the tags of all the predecessors of a current state and selects the tag associated with the probability of the state that maximizes eq.(1). In order to do that the Backtrace Processor gets the according information from the Viterbi Processor.

In order to minimize the backtrace memory requirements only the backtrace tags associated with high probabilities are stored in the backtrace memory. This pruning operation is performed by the Backtrace Memory Processor.

4 PROCESSOR DESCRIPTIONS

This section gives a description of the three custom processors used in the subsystem, the Viterbi Processor, the Backtrace Processor and the Backtrace Memory Processor (see Fig.3). Our implementation uses a logarithmic representation of probabilities thus reducing multiplications to additions and changing max operations to min operations. This realization

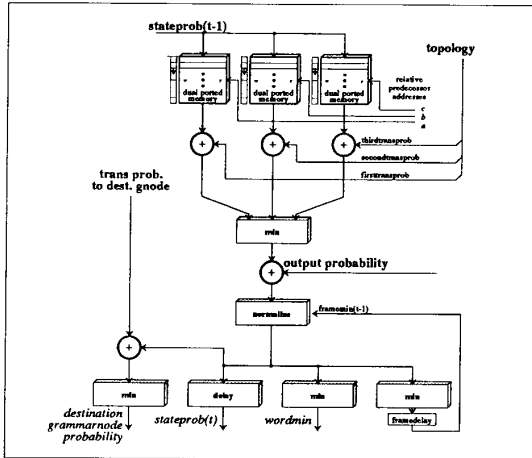


Figure 4: Architecture of the Viterbi Processor.

is feasible since the logarithmic probabilities of $a_{i,j}$ and $b_{j,k}$, stored in the topology memory and in the output probability memory, are precompiled.

4.1 Viterbi Processor

The task of the Viterbi Processor is to update the state probabilities according to eq.(1). Fig.4 shows a block diagram of the processor architecture.

The states of the recognition vocabulary are sequentially processed which conforms to the sequential access of the memories in the system. To speed up the computation the processing of 3 predecessors is done in parallel. Also, the computation is heavily pipelined so that for 3 predecessors the evaluation of eq.(1) can be completed within 1 processor cycle.

The on-chip memories that store the state probabilities at time $t-1$ are implemented as bidirectional memories. This allows a simultaneous read and write operation where the write address is sequentially incremented and the read address is random (see section 3). The three read addresses are generated by adding an offset to the current write address (which corresponds to the address of the current state). Thus, 3 offset addresses corresponding to the three predecessor locations relative to the current state along with the associated transition probabilities are read from the external topology memory.

The outputs of the bidirectional memories correspond to three state probabilities $Prob(O_{t-1}, s_i|M)$ at time $t-1$ of three predecessors of a given state. According to eq.(1) these probabilities have to be multiplied with the associated transition probabilities to the current state, $a_{i,j}$, implemented as an addition of the logarithm of the probabilities.

After that operation, the probability with the highest value

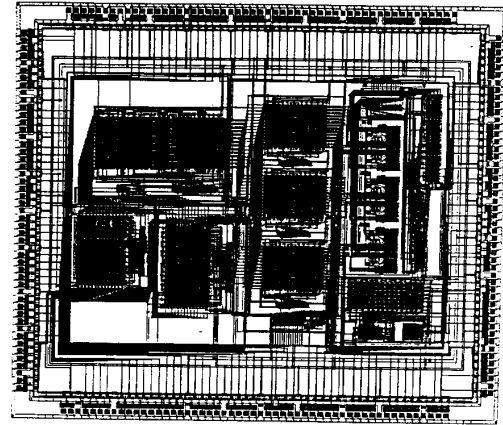


Figure 5: Layout of the Viterbi Processor.

is selected corresponding to a minimum operation of the logarithmic values. Whenever a state has more than 3 predecessors, this minimum operation is done sequentially on sets of 3 predecessor probabilities to find out the overall best value. To this selected value the output probability $b_{j,k}$, read from the off chip output probability memory, is multiplied (added) to finally compute $Prob(O_t, s_j|M)$.

In order to minimize the wordlength of the probability values the processor performs a normalization based on the overall best probability of the previous frame. Also, to support a pruning operation on the word level for later implementations, the highest state probability of every word is computed.

Parallel to the computation of (1) the Viterbi Processor computes a destination grammar node probability. This probability is the overall maximum of the state probabilities of a given word multiplied (added) with the associated transition probabilities. The implementation is such that potentially every node can end a word, but only meaningful transitions have a non-zero transition probability associated with the destination grammar node transition.

The processor (Fig.5) has up to 13 levels of pipelining. The basic blocks are 8 individual datapaths, 3 bidirectional memories and a controller unit. The chip was fabricated in a $2\ \mu\text{m}$ technology. Including 204 pads, it has a die size of $11.6 \times 9.8\text{mm}^2$ with 25,000 transistors.

4.2 Backtrace Processor

The Backtrace Processor implements eq.(3), the backtrace portion of the Viterbi algorithm. Like the Viterbi Processor, this processor has 3 on-chip bidirectional memories to store the backtrace tags of 3 predecessors of a given state.

The tags are passed through a series of delay registers to synchronize the data to the associated probability data that

are simultaneously processed on the Viterbi Processor. Multiplexors are used to generate two different outputs based on control signals that are supplied by the Viterbi Processor. The first output is the updated backtrace tag for every state within a word, and is stored in the state probability memory. It is the copy of the tag of the predecessor state from the previous frame. The second output is the backtrace information of the destination grammar node. It is either stored in the backtrace memory to recover the word sequence after the end of a sentence or, if the probability of this grammar node is beyond a threshold value, it is passed to the grammar processor.

The backtrace processor has a die size of 6.8 mm by 7.5 mm including 132 pads in a 2 micron technology and uses 12,000 transistors.

4.3 Backtrace Memory Processor

Since there is no end-of-word detection, each grammar node would need an entry in the backtrace memory for every time frame to implement a linked list of grammar nodes. That means, the stored predecessor of a grammar node is in most cases the grammar node itself.

To minimize backtrace information the Backtrace Memory Processor only stores a grammar node in the Backtrace Memory if the probability associated with this grammar node is higher than a threshold probability. The processor also computes this pruning threshold based on a running maximum probability. The running maximum can be preset at the start of each frame to avoid initialization effects.

The processor is a slave processor to the Viterbi Processor. It is not driven by the system clock as it only requires a strobe generated by the Viterbi Processor to start the pruning and threshold updating operations after a word has been processed.

Due to this asynchronous behavior and due to the low computational rate (once for each grammar node) a standard cell design methodology is used. The chip has a die size of 6.8 mm by 6.8 mm including 132 pads.

5 CONCLUSION

The architecture and VLSI implementation of a word processing subsystem for a real time speech recognition system using Hidden Markov Models has been described. All the custom processors were designed using the LagerIV Silicon Assembly System. The description of the chips is textual and parametrized allowing for easy redesigns in future implementations, especially for changes such as different wordlengths for probabilities or backtrace tags.

Acknowledgement

This project was funded by DARPA under Grant 25815. The authors wish to thank all the people at BBN, SRI and UC Berkeley that contributed to this project. In particular we want to thank Dr. H. Murveit of SRI and Dr. R. Schwartz of BBN for their invaluable contributions. The first author also wishes to thank Prof. Pfeleiderer acting for the Siemens Research Laboratories in West Germany for his support.

References

- [1] A. Averbuch et al. *An IBM-PC based Large-Vocabulary Isolated-Utterance Speech Recognizer*, in Proc. ICASSP 86: 1986 International Conference on Acoustics Speech and Signal Processing, pages 53–56, April 1986.
- [2] 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*, In Proc ICASSP 87: 1987 International Conference on Acoustics Speech and Signal Processing, pages 89–92, April 1987.
- [3] W. Drews, R. Laroia, J. Pandel, A. Schumacher, and A. Stölzle. *A CMOS Processor for a 1000 Word Speech Recognition System*, in Proceedings of the IEEE Custom Integrated Circuits Conference, pages 559–562, May 1987.
- [4] Robert A Kavalier. *The Design and Evaluation of a Speech Recognition System for Engineering Wordstations*. PhD thesis, University of California at Berkeley, May 1986.
- [5] K.Lee and H. Hsiao-Wuen. *Large Vocabulary Speaker-Independent Continuous-Speech Recognition using HMM*, in Proc. ICASSP 88: 1988 International Conference on Acoustics Speech and Signal Processing, pages 123–126, April 1988.
- [6] H. Murveit and M. Weintraub. *1000 Word Speaker Independent Continuous Speech Recognition System using Hidden Markov Models*, In Proc ICASSP 87: 1987 International Conference on Acoustics Speech and Signal Processing, pages 115–118, April 1988.
- [7] J. Rabaey, R. Brodersen, A. Stölzle, S. Narayanaswamy, D. Chen, R. Yu, P. Schrupp, H. Murveit, and A. Santos. *VLSI Signal Processing III. chapter A Large Vocabulary Real Time Continuous Speech Recognition System*, pages 61–74. IEEE Press, 1988.
- [8] L R Rabiner and B H Juang. *An Introduction to Hidden Markov Models*, IEEE ASSP Magazine, pp. 4–16, January 1986.