

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

D. C. Chen, R. Yu, J. Rabaey, R. W. Brodersen
University of California at Berkeley, Dept. of EECS

Abstract

This paper summarizes the architecture and implementation of a grammar processing subsystem for a large vocabulary, real time, continuous speech recognition system. This subsystem contains two custom VLSI chips which perform the evaluation of starting word probabilities associated with the across-word transitions in the hidden Markov model based (HMM) speech recognition system. This system has a maximum computation rate of 200 MOPS and an IO bandwidth of 265 MByte/sec.

1 INTRODUCTION

The hidden Markov model based algorithms are currently the most effective techniques used in speech recognition systems [7, 2, 5]. Our system [6, 4] uses HMM algorithms along with language models to recognize, in real time, continuous speech composed of a 3000 word vocabulary. It will be able to run a class of HMM based systems being developed at SRI, BBN and CMU [1].

As shown in Figure 1, the system is logically and computationally partitioned into two subsystems. The **word processing subsystem**, which performs the Viterbi algorithm to find the probabilities of states within individual words and also the probability that a word ends, was presented in CICC 1989 [8]. Speech recognition accuracy is further enhanced when syntactic constraints are imposed on the concatenation of individual words in the vocabulary. The **grammar**

processing subsystem searches for the most probable word sequence given transition probabilities in this end-of-word to start-of-word, or bigram, model.

The grammar processing subsystem combines the acoustic word sequence probabilities, generated by the word processing subsystem, with pre-compiled word sequence probabilities generated by a grammar model of the English language. The model currently supported is a statistical grammar with transitions between all words. The recognized sentence is the sentence with the highest combined acoustic and language model probabilities.

This paper describes the algorithm, architecture, and implementation of the grammar subsystem.

2 ALGORITHM

Within each time frame of 10 msec, the grammar subsystem evaluates the probabilities that words in the vocabulary start and keeps pointers to the most probable path leading to that word. These pointers are used to reconstruct, or backtrace, the most probable word sequence upon sentence completion. The algorithms used to evaluate the word probabilities are based on two models: The statistical grammar model and the ϵ -model. The statistical model is used to handle word arcs with high probabilities, and the ϵ -model word arcs with low probabilities.

2.1 Statistical-Model

The statistical grammar model allows any word to follow any other word. Associated with the i th word outputted by the word processing subsystem is a probability PGO_i , the probability that a word i ends at a particular point. The grammar subsystem then calculates a probability PGI_j , the probability that word j starts in the next frame. This j th successor word probability is then sent back to the word processing subsystem. The probability $PGI_j^G(t+1)$ under the statistical grammar model is found by using:

$$PGI_j^G(t+1) = \max_{i \in \text{all words}} [PGO_i(t) \times c_{ij}] \quad (1)$$

where c_{ij} is the transition probability from word i to word j . Figure 2 shows a graph of the statistical grammar model. The evaluation in equation (1) is terminated when the probability falls below a variable threshold.

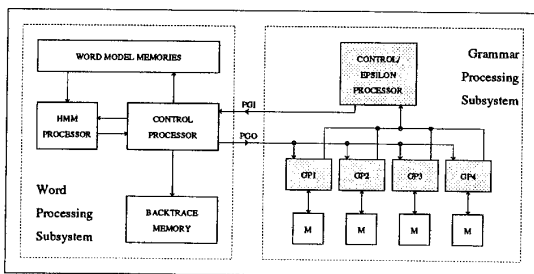


Figure 1: Block diagram of the word processing and grammar processing subsystems.

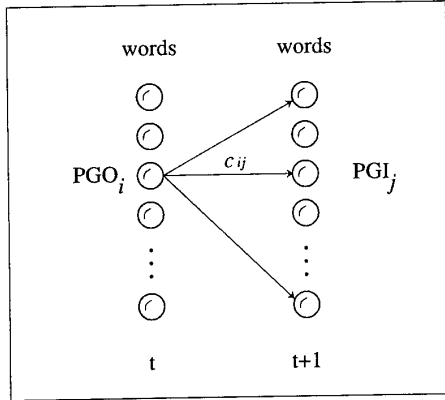


Figure 2: Graph of evaluation using the statistical grammar model.

2.2 ϵ -Model

For a vocabulary of 3000 words, a fully connected graph would require 9,000,000 transition evaluations and storage locations. To alleviate the excessive computational and storage requirements associated with modeling a fully connected graph, we have adopted a simplified model, called the ϵ -model, to handle word arcs with low probabilities. We shall see later how storage requirements are reduced from N^2 to $2N$ for a vocabulary size N by using the ϵ -model.

In this model, the low probabilities c_{ij} are approximated by probability ϵ_{ij} , where $\epsilon_{ij} = \epsilon_i \times \epsilon_j$. ϵ_i represents a probability **out of** the i th word, and ϵ_j represents a probability **into** the j th word. The probability $PGI_j^t(t+1)$ calculated under the ϵ -model is found similarly:

$$PGI_j^t(t+1) = \max_{i \in \text{all words}} [PGO_i(t) \times \epsilon_i] \times \epsilon_j \quad (2)$$

The successor probability sent to the word processing subsystem is the larger of the word probabilities calculated under equations (1) and (2):

$$PGI_j(t+1) = \max[PGI_j^G(t+1), PGI_j^t(t+1)] \quad (3)$$

Associated with the successor word probability PGI_j is a backtrace pointer to the most probable predecessor word.

3 SUBSYSTEM ARCHITECTURE

Figure 3 shows a high level block diagram of the grammar processing subsystem. The Grammar Processor implements equation (1), and the Epsilon Processor implements equations (2) and (3). That is, the Grammar Processor handles the subset of word arcs with high probabilities, and the Epsilon Processor handles the low probabilities.

3.1 Task Partitioning

To meet the computational requirements in the statistical grammar model, we partitioned the task into several Gram-

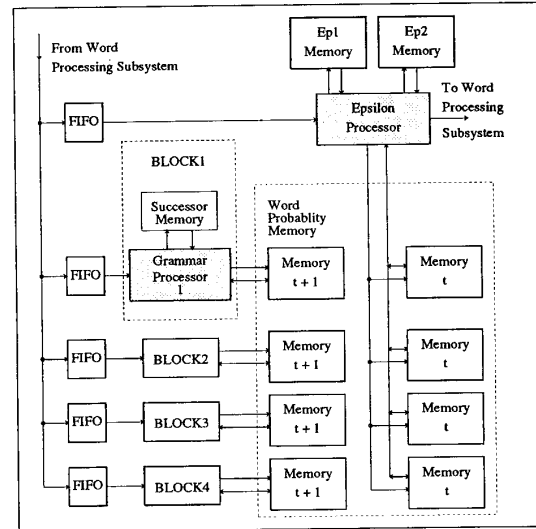


Figure 3: Block diagram of grammar processing subsystem.

mar Processors. With a 5 MHz system clock, a single Grammar Processor can update 50,000 starting word probabilities in the 10 msec time frame. This corresponds to an average of 17 successors per word, given a 3000 word vocabulary.

Our initial configuration will have four Grammar Processors working in parallel to update an equivalent 200,000 starting word probabilities, or an average of 70 successors per word. As shown in Figure 4, each Grammar Processor updates a subset of the total Word Probability Memory. With this partitioning, memory contention among Grammar Processors is not present and duplication of memory is not required.

It is possible to add even more Grammar Processors to handle systems with larger grammars and/or to increase the overall throughput.

3.2 Memories

3.2.1 FIFO's

The word and grammar processing subsystems communicate through several FIFO's, which buffer the word probabilities (PGO) and their associated backtrace pointers. Each processor monitors and accesses its own FIFO, allowing for independent and concurrent operation.

3.2.2 Epsilon Memories

The probabilities ϵ_i and ϵ_j associated with each word are stored in the Ep1 and Ep2 Memories, respectively, and are accessed by the Epsilon Processor.

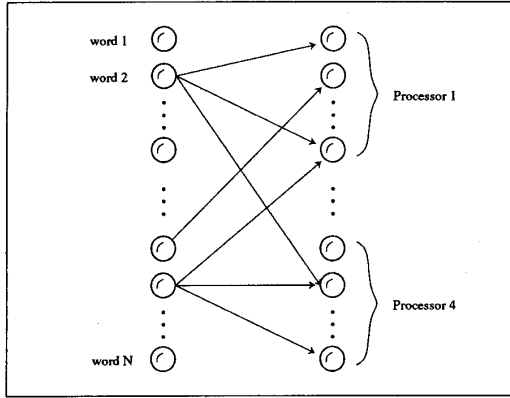


Figure 4: Partitioning of the task of updating word probabilities.

3.2.3 Successor Memories

Each Successor Memory is accessed by a separate Grammar Processor and contains the lists of successor words handled by that Grammar Processor. An entry in the Successor Memory contains three fields: The first is an address of the successor in the Word Probability Memory; the second is a transition probability c_{ij} to that successor; and the third is a flag indicating the end of the current successor list.

3.2.4 Word Probability Memory

The Word Probability Memory consists of two banks corresponding to the starting word probabilities (PGI) at the current and next time frames. Each bank is further subdivided into four groups, representing the successor word partitioning. During a given time frame, the "current" bank is accessed by the Epsilon Processor, and the "next" bank is accessed separately by the four Grammar Processors. The access of these banks is swapped with each successive time frame.

An entry in the Word Probability Memory contains two fields: The word probability itself and the associated backtrace pointer to the predecessor word.

3.3 Operation

The grammar processing subsystem operates as follows. In a given time frame, the Epsilon Processor takes the maximum value calculated in the previous frame, and multiplies it with the ε_j value from the Ep2 Memory to generate the j th word probability $PGI_j^G(t)$ under the ε -model, corresponding to equation (2).

This probability is then compared with the word probability found in the "current" bank of the Word Probability Memory, and the larger of the two is sent to the word processing subsystem, corresponding to equation (3). In the same time frame, the word processing subsystem returns the new word probabilities $PGO_i(t)$ based on acoustic information.

These probabilities and backtrace values are pushed into all the receiving FIFO's in the grammar processing subsystem.

For the i th incoming word, a Grammar Processor looks into the Successor Memory to find the pre-compiled list of successor words and their transition probabilities c_{ij} 's. The probability $PGI_j^G(t+1)$ of the j th successor word is next fetched from the Word Probability Memory, and compared to the product of $PGO_i(t)$ and c_{ij} . The larger of the two is written back into the Word Probability Memory as $PGI_j^G(t+1)$. This sequence of operations implements equation (1).

In the same frame, the Epsilon Processor keeps running maximum of the product of the incoming word probabilities $PGO_i(t)$ and ε_i values, which are stored in the Ep1 Memory. This operation corresponds to the maximizing part of equation (2).

The grammar subsystem signals completion when all the words have been sent and the Grammar Processors have all completed successor updating. The two banks of the Word Probability Memories are swapped in the next time frame, and our description of operation is complete.

4 PROCESSOR IMPLEMENTATION

To implement the architecture summarized above, we designed two custom VLSI chips using the LAGERIV CAD system [3] and manufactured them in $2\ \mu\text{m}$ CMOS technology through the MOSIS facility. This section presents some features of these custom chips.

In both of these chips, arithmetic operations are done in the log domain so that adders instead of multipliers can be used, thereby conserving chip area. Although we partitioned the functionality to minimize chip I/O requirements, both chips remain pin limited.

4.1 Grammar Processor

Figure 5 shows a simplified logic block diagram of the Grammar Processor. Sections include address generation, control, threshold calculation, and a five-stage pipelined datapath (not shown) to perform equation (1). By performing read and write into different locations in the Word Probability Memory within a 200 ns cycle, a single Grammar Processor can process approximately 50,000 word arcs in the 10 msec frame.

A dynamic thresholding mechanism allows the Grammar Processor to stop processing the remaining successors of a given word once the probability of the successors falls below the threshold. The transition probabilities c_{ij} in the Successor Memories are stored in decreasing order, so that we can easily detect when this threshold is crossed. Within each frame, the threshold probability is computed using a running maximum probability scaled by a user-specified offset factor.

The Grammar processor has 142 signal pins, 11,385 transistors, and a die size of $9.7 \times 10.5\ \text{mm}^2$. Figure 6 shows the layout of the processor.

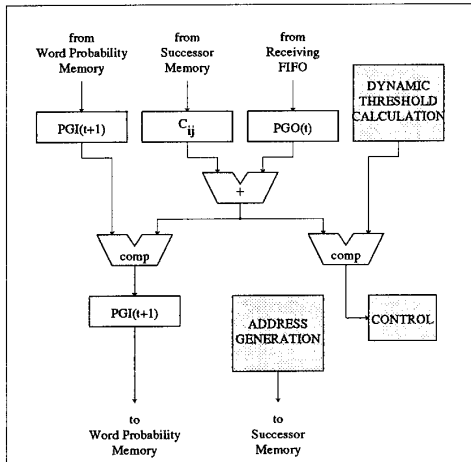


Figure 5: Simplified block diagram of Grammar Processor.

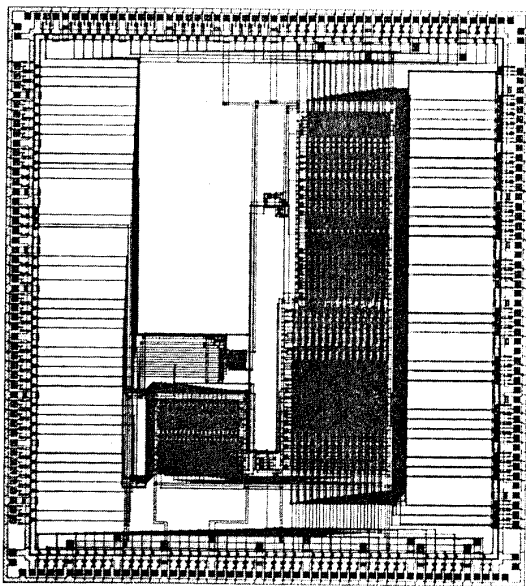


Figure 6: Layout of the Grammar Processor.

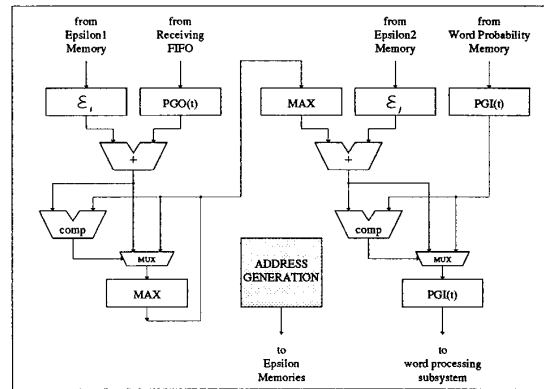


Figure 7: Simplified block diagram of Epsilon Processor.

4.2 Epsilon Processor

The Epsilon Processor contains two independent sections, each with its own controller and datapaths. Figure 7 shows a simplified logic block diagram of the Epsilon Processor. One section calculates the running maximum over the product of each word probability and its associated ϵ_i value. The other section calculates the product of this maximum value and the associated ϵ_j value and compares it to the word probabilities. The Epsilon Processor sends the maximum between these two values to the word processing subsystem according to equation (3). It also signals completion after all the words in the vocabulary have been sent to the word processing subsystem and all the Grammar Processors are finished with updating successor probabilities.

The Epsilon Processor has 157 signal pins, 10,775 transistors, and a die size of $9.8 \times 9.7 \text{ mm}^2$. Figure 8 shows the layout of the processor.

5 CONCLUSION

This paper has summarized the architecture and custom implementation of a grammar processing subsystem for a real time large vocabulary continuous speech recognition system using hidden Markov models. Our prototype has 3000 words, and larger vocabularies and higher throughput can be obtained by scaling the subsystem.

Acknowledgement

This project was funded by DARPA. The authors wish to thank H. Murveit, R. Schwartz, A. Santos, and others at SRI and UC Berkeley who have contributed to this project.

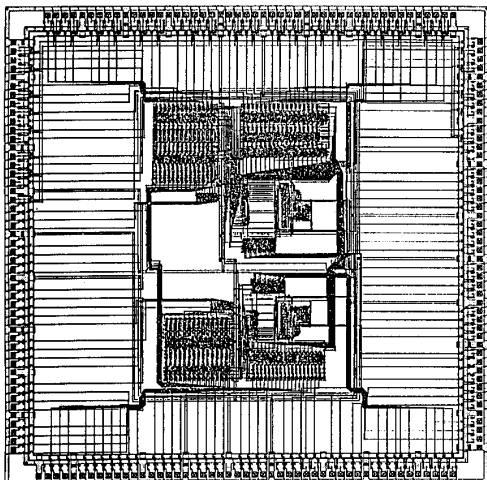


Figure 8: Layout of the Epsilon Processor.

References

- [1] 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.
- [2] K.Lee and H. Hsiao-Wuen. Large vocabulary speaker-independent continuous-speech recognition using lhm. In *Proc. ICASSP 88: 1988 International Conference on Acoustics Speech and Signal Processing*, pages 123–126, April 1988.
- [3] Electronic Research Laboratory. *LagerIV Distribution 1.0 Silicon Assembly System Manual*. University of California at Berkeley, June 1988. Distribution 1.0.
- [4] H. Murveit, J. Mankoski, J. Rabaey, R. Brodersen, A. Stölzle, S. Narayanaswamy, D. Chen, R. Yu, P. Schrupp, H. Murveit, and A. Santos. A large-vocabulary real-time continuous-speech recognition system. In *Proc ICASSP 89: 1989 International Conference on Acoustics Speech and Signal Processing*, April 1989.
- [5] 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.
- [6] 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.
- [7] L R Rabiner and B H Juang. An introduction to hidden markov models. *IEEE ASSP Magazine*, pages 4–16, January 1986.
- [8] A. Stölzle, S. Narayanaswamy, K.Kornegay, R. W. Brodersen, and J. Rabaey. A vlsi wordprocessing subsystem for a real time large vocabulary speech recognition system. In *Proc. CICC'89: 1989 Custom Integrated Circuits Conference*, May 1989.