Toward a Real-Time Spoken Language System Using Commercial Hardware

Steve Austin, Pat Peterson, Paul Placeway, Richard Schwartz, Jeff Vandergrift

BBN Systems and Technologies Inc.
10 Moulton St.
Cambridge, MA, 02138

Abstract

We describe the methods and hardware that we are using to produce a real-time demonstration of an integrated Spoken Language System. We describe algorithms that greatly reduce the computation needed to compute the N-Best sentence hypotheses. To avoid grammar coverage problems we use a fully-connected first-order statistical class grammar. The speech-search algorithm is implemented on a board with a single Intel i860 chip, which provides a factor of 5 speedup over a SUN 4 for straight C code. The board plugs directly into the VME bus of the SUN4, which controls the system and contains the natural language system and application back end.

1. Introduction

One goal of the Spoken Language System (SLS) project is to demonstrate a real-time interactive system that integrates speech recognition and natural language processing. We believe that, currently, the most practical and efficient way to integrate speech recognition with natural language is using the N-Best paradigm, in which we find the most likely whole-sentence hypotheses using speech recognition and a simple language model, and then filter and reorder these hypotheses with natural language. Although we claim that this process is efficient, finding the N-Best sentences still requires significant amounts of computation.

To accelerate the speech recognition process, several sites have been developing special-purpose hardware, from fully-custom VLSI [5] to custom boards based on general-purpose processors operating in parallel [1]. However, with the rapid changes in commercial hardware in the past few years we felt that if we *could* achieve real time with commercially available hardware we would realize a number of advantages, such as decreased development time, increased reliability, and better hardware and software support. We would not have to divert attention from speech-recognition research and our results would be more easily shared with the community. We felt that these hopes were not unrealistic because, for our algorithms in particular, which are research-oriented and seldom have a stable, regular structure, the potential gains from custom hardware would not be that great.

Our strategy in developing a real-time spoken language system has been to find appropriate commercial hardware and then tailor our algorithms to the hardware and application. In this context, we are trying to approach the following goals:

1. real-time processing with a short delay
2. reasonable-cost, commercially-available hardware
3. source code compatibility with our research programs
4. computation of the N-Best sentences for large N
5. use of a robust fully-connected statistical grammar
6. practical memory requirements
7. negligible loss of accuracy
8. ability to handle large vocabulary

Two of the new algorithms used in this effort are described in a separate paper [8]. Specifically, the Word-Dependent N-Best search and the Forward-Backward Search. In Section 2 we describe the class of hardware that we have chosen for this task. Section 3 reviews the reasons for using a statistical grammar and presents a new more efficient algorithm for time-synchronous decoding with a statistical grammar. In Section 4 we compare the accuracy of the standard Viterbi algorithm with the time-synchronous forward search algorithm that we use. And in Section 5 we give the latest status of the speed and accuracy of our system.

2. Hardware

It is already quite straightforward to perform signal processing in real-time on current boards with signal processor chips. However, the speech recognition search requires a large amount of computation together with several MB of fast readily accessible memory. In the past there have not been commercially available boards or inexpensive computers that meet these needs. However this is changing. The Motorola 88000 and Intel 860 chips are being put on boards with substantial amounts of random access memory. Most chips now come with C compilers, which means that the bulk of development programs can be transfered directly. If needed, computationally intensive inner loops can be hand coded.

After considering several choices we have chosen boards based on the Intel 860 processor. The Intel 860 processor combines a RISC core and a floating-point processor with a peak speed of 80 MFLOPS. Currently, we have looked at VME boards made by Sky Computer (the SkyBolt) and Mercury (the MC860). The SkyBolt currently is available with 4 MB of static RAM. It will very shortly be available with 16 MB of DRAM and 256 KB of static RAM cache. The Mercury MC860 is currently available with 16 MB of DRAM. Most C programs that we have run on both of these machines run about five times faster than on a SUN 4/280.

Figure 1 illustrates the hardware configuration that we have built. The host will be a SUN 4/330. The microphone is connected to an external preamp and A/D converter which connects directly to the serial port of the Sky Challenger. The Sky Challenger with dual TMS320C30s will be used for signal processing and vector quantization (VQ). The SkyBolt will be used for the speech recognition N-Best search. The boards communicate with the host and each other through the VME bus, making high speed data transfers easy. However currently the data transfer rate between the boards is very low. The SUN 4 will control the overall system and will also contain the natural language understanding system and the application back end.

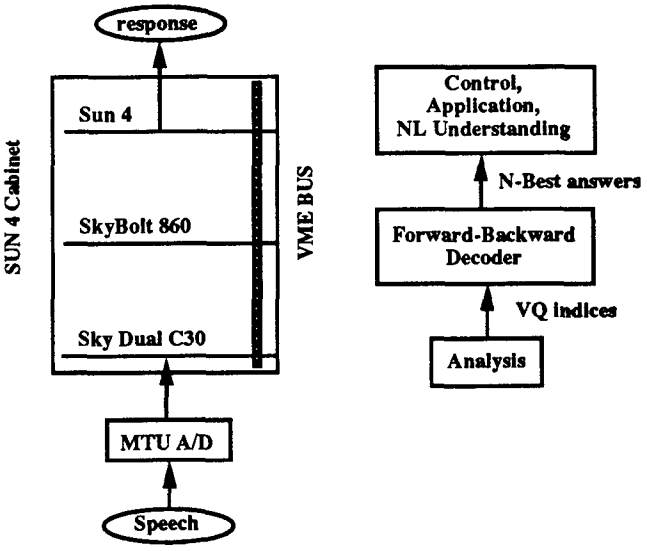

Figure 1: Real-Time Hardware Configuration. The Sky Challenger Dual C30 board and the Intel 860 board plug directly into the VME bus of the SUN 4.

We use all three processors during most of the computation. When speech has started the C30 board will compute the signal processing and VQ in real-time. The SUN 4 will accumulate the speech for possible long-term storage or playback. Meanwhile, the Intel 860 will compute the forward pass of the forward-backward search. When the end of the utterance has been detected, the SUN will give the 1-Best answer to the natural language understanding system for parsing and interpretation. Meanwhile the Intel 860 will search backwards for the remainder of the N Best sentence hypotheses. These should be completed in about the same time that the NL system requires to parse the first answer. Then, the NL system can parse down the list of alternative sentences until an acceptable sentence is found.

Currently, the computation required for parsing each sentence hypothesis is about 1/2 second. The delay for the N-Best search is about half the duration of the sentence. This is expected to decrease with further algorithm improvements.

2. Time-Synchronous Statistical Language Model Search

We know that any language model that severely limits what sentences are legal cannot be used in a real SLS because people will almost always violate the constraints of the language model. Thus, a Word-Pair type language model will have a fixed high error rate. The group at IBM has long been an advocate of statistical language models that can reduce the entropy or perplexity of the language while still allowing all possible word sequences with some probability. For most SLS domains where there is not a large amount of training data available, it is most practical to use a statistical model of word classes rather than individual words. We have circulated a so called Class Grammar for the Resource Management Domain [3]. The language model was simply constructed, having only first-order statistics and not distinguishing the probability of different words within a class. The measured test set perplexity of this language model is about 100. While more powerful "fair" models could be constructed, we felt that this model would predict the difficulty of a somewhat larger task domain. The word error rate is typically twice that of the Word-Pair (WP) grammar. One problem with this type of grammar is that the computation is quite a bit larger than for the WP grammar, since all 1000 words can follow each word (rather than an average of 60 as in the WP grammar).

During our work on statistical grammars in 1987 [6], we developed a technique that would greatly reduce the computational cost for a time-synchronous search with a statistical grammar[1] . Figure 2 illustrates a fully-connected first-order statistical grammar. If the number of classes is C, then the number of null-arcs connecting the nodes is C^2. However, since the language models are rarely well-estimated, most of the class pairs are never observed in the training data. Therefore, most of these null-arc transition probabilities are estimated indirectly. Two simple techniques that are commonly used are padding, or interpolating with a lower order model. In padding we assume that we have seen every pair of words or classes once before we start training. Thus we estimate $p(c_2|c_1)$ as

$$p(c_2|c_1) = \frac{N(c_1, c_2) + 1}{N(c_1) + C}$$

[1]We should note that we have heard that this algorithm was independently arrived at by Andres Santos from the University of Madrid while on sabbatical at SRI in 1989.

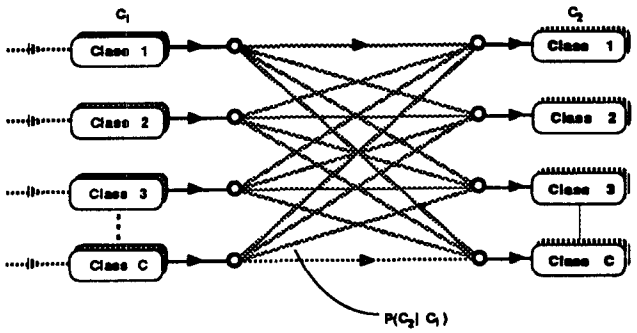

Figure 2: Fully Connected First-Order Statistical Grammar. Requires C^2 null arcs.

In interpolation we average the first-order probability with the zeroth-order probability with a weight that depends on the number of occurrences of the first class.

$$p(c_2|c_1) = \lambda(c_1)\hat{p}(c_2|c1) + [1 - \lambda(c_1)]\hat{p}(c_2)$$

where

$$\hat{p}(c_2|c1) = \frac{N(c_1, c_2)}{N(c_1)}$$

and

$$\hat{p}(c_2) = \frac{N(c_2)}{N(all\ words)}$$

In either case, when the pair of classes has never occurred, the probability can be represented much more simply. For the latter case of interpolated models, when $N(c_1, c_2) = 0$ the expression simplifies to just

$$[1 - \lambda(c_1)]\hat{p}(c_2)$$

The first term, $1 - \lambda(c_1)$, depends only on the first class, while the second term, $\hat{p}(c_2)$, depends only on the second class. We can represent all of these probabilities by adding a zero-order state to the language model. Figure 3 illustrates this model. From each class node we have a null transition to the zero-order state with a probability given by the first term. Then, from the zero-order state to each of the following class nodes we have the zero-order probability of that class.

Now that the probabilities for all of the estimated transitions has been taken care of we only need the null transitions that have probabilities estimated from actual occurrences of the pairs of classes, as shown in Figure 4. Assuming that, on average, there are B different classes that were observed to follow each class, where $B << C$, the total number of transitions is only $C(B + 2)$. For the 100-class grammar we find that $B = 14.8$, so we have 1680 transitions instead of 10,000. This savings reduces both the computation and storage associated with using a statistical grammar.

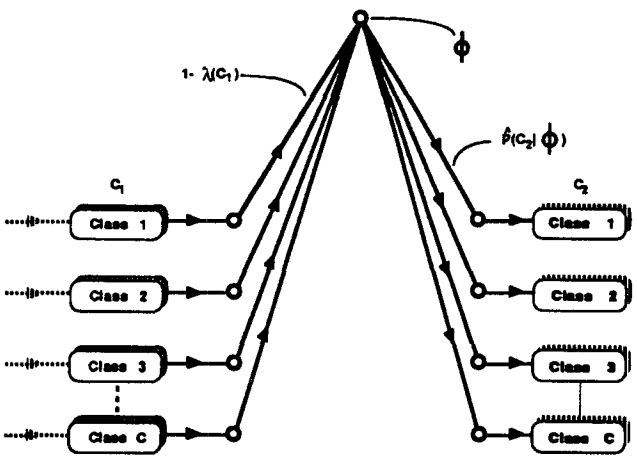

Figure 3: Zero-state within first-order statistical grammar. All of the transitions estimated from no data are modeled by transitions to and from the zero-state.

It should be clear that this technique can easily be extended to a higher order language model. The unobserved second-order transitions would be removed and replaced with transitions to a general first-order state for each word or class. From these we then have first-order probabilities to each of the following words or classes. As we increase the order of the language model, the percentage of transitions that are estimated only from lower order occurrences is expected to increase. Thus, the relative savings by using this algorithm will increase.

3. Time-synchronous Forward Search vs Viterbi

The search algorithm that is most commonly used is the Viterbi algorithm. This algorithm has nice properties in that it can proceed in real time in a time-synchronous manner, is quite amenable to the beam-search pruning algorithm [4], and is also relatively easy to implement on a parallel processor. Another advantage is that it only requires compares and adds (if we use log probabilities). Unfortunately, the Viterbi algorithm finds the most likely sequence of states rather than the most likely sequence of words.

To correctly compute the probability of any particular sequence of words requires that we add the probabilities of all possible state sequences for those words. This can be done with the "forward pass" of the forward-backward training algorithm. The only difference between the Viterbi scoring and the Forward-pass computation is that we add the probabilities of different theories coming to a state rather than taking the maximum.

We presented a search algorithm in 1985 [7] that embodied most of this effect. Basically, within words we add probabilities, while between words we take the maximum. It was not proven at that time how much better, if any, this algorithm was than the simpler Viterbi algorithm, and whether it was as good as the strictly correct algorithm that computes

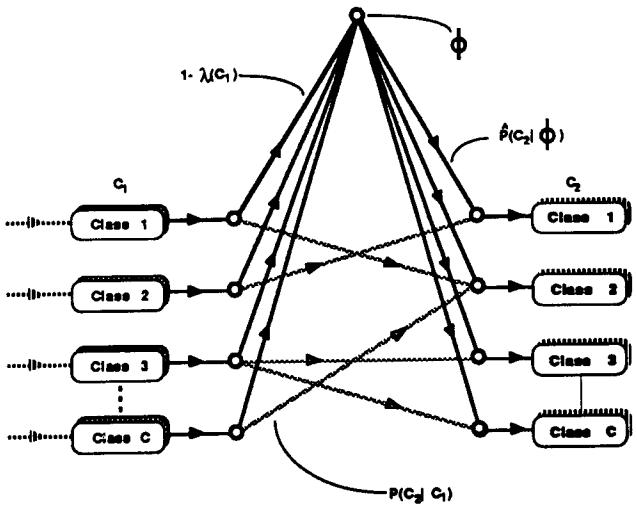

Figure 4: Sparsely Connected First-Order Statistical Grammar with zero-state requires many fewer null arcs.

the score of each hypothesis independently.

When we compared these two algorithms under several conditions, we found that there was a consistent advantage for adding the probabilities within the word. For example, when we use the class grammar, we find that the word error rate decreases from 8% to 6%.

To be sure that the time-synchronous forward search gives us the same performance as the ideal forward score is somewhat more complicated. We must guarantee that we have found the highest scoring sentence with the true forward probability score. One way to find this is to use the exact N-Best algorithm [2]. Since the exact N-Best algorithm separates the computation for any two different hypotheses, the scores that result are, in fact, the correct forward probabilities, as long as we set N to a large enough value. A second, much simpler way to verify the time-synchronous algorithm is to see if it ever gets a wrong answer that scores worse than the correct answer. We ran a test in which all incorrect answers were rescored individually using the forward probability. We compared these scores to the forward probability for the correct answer. In no case (out of 300 sentences) did the time-synchronous forward search ever produce a wrong answer that, in fact, scored worse than the correct answer.

The reason that this whole discussion about the Viterbi algorithm is relevant here is that the Viterbi algorithm is faster than the forward search. Therefore, we use the integer Viterbi algorithm in the forward-pass of the Forward-Backward Search. Since the function of the forward-pass is primarily to say which words are likely, it is not essential that we get the best possible answer. The backward N-Best search is then done using the better-performing algorithm that adds different state-sequence probabilities for the same word sequence.

4. Speed and Accuracy

When we started this effort in January, 1990, our unoptimized time-synchronous forward search algorithm took about 30 times real time for recognition with the WP grammar and a beamwidth set to avoid pruning errors. The class grammar required 10 times more computation. The exact N-Best algorithm required about 3,000 times real time to find the best 20 answers. When we required the best 100 answers, the program required about 10,000 times real time. Since January we have implemented several algorithms, optimized the code, and used the Intel 860 board to speed up the processing. The N-Best pass now runs in about 1/2 real time. Below we give each of these methods along with the factor of speed gained.

Statistical grammar algorithm	5
Word-Dependent N-Best	5
Forward-Backward Search	40
Code Optimization	4
Intel 860 Board	5
Total reduction in computation	20,000

As can be seen, the three algorithmic changes accounted for a factor of 1,000, while the code optimization and faster processor accounted for a factor of 20. We expect any additional large factors in speed to come from algorithmic changes. When the VLSI HMM processor becomes available, the speed of the HMM part of the problem will increase considerably, and the bottleneck will be in the language model processor. We estimate that the language model computation accounts for about one third of the total computation.

Our current plan is to increase the speed as necessary and complete the integration with the natural language understanding and application backend by September, 1990.

Accuracy

It is relatively easy to achieve real time if we relax our goals for accuracy. For example, we could simply reduce the pruning beamwidth in the beam search and we know that the program speeds up tremendously. However, if we reduce the beamwidth too much, we begin to incur search errors. That is, the answer that we find is not, in fact, the highest scoring answer. There are also several algorithms that we could use that require less computation but increase the error rate. While some tradeoffs are reasonable, it is important that any discussion of real-time computation be accompanied by a statement of the accuracy relative to the best possible conditions.

In Table 1 below we show the recognition accuracy results under several different conditions. All results use speaker-dependent models and are tested on the 300 sentences in the June '88 test set. For each condition we state whether the forward pass would run in less than real time on the SkyBolt for more than 80% of the sentences — which is basically a function of the pruning beamwidth. The backward pass currently runs in less than 1/2 real time, and we expect it will get faster. We don't yet have a good feeling for how much delay will be tolerable, but our goal is for the delay in

computing the N Best sentences to be shorter than the time needed for natural language to process the first sentence, or about 1/2 second. The accuracy runs were done on the SUN 4/280. Based on our speed measurements, we assume that anything that runs in under five times real-time on the SUN 4 will run in real-time on the Intel 860 board. For a similar CG condition whose forward pass ran in under five times real-time on the SUN 4, we verified real-time operation on a 4 MB SkyBolt.

Grammar	RT?	Word Err	1 Best	20 Best	100 Best
WP-XW	N	1.9			
WP	N	3.9	19.7	2.3	2.0
WP	Y	3.9	20.0	2.7	2.7
CG-XW	N	4.7			
CG	N	8.2	38.7	7.0	4.0
CG	1.2	8.5	39.3	8.7	5.7
CG	Y	9.1	40.0	11.7	9.3

Table 1: Word and sentence error rates for the real-time N-Best algorithm compared with the best non-real-time conditions.

For each condition we give the word error, 1-Best sentence error, and N-Best sentence error for N of 20 and 100. "N-Best sentence error" Results are given for the Word-Pair (WP) grammar and for the Class (CG) Grammar. The conditions WP-XW and CG-XW were done using cross-word triphone models that span across words and have been smoothed with the triphone cooccurence smoothing. These conditions were only decoded with the 1-Best forward-search algorithm, and so produced only word error statistics for reference. The models that do not use cross-word triphones also do not use triphone cooccurence smoothing. Since the forward pass is done using the Viterbi algorithm, this affects the word error rate and the 1-Best sentence error rate, which are measured from the forward pass only.

Currently we have not run the cross-word models with the N-Best algorithm. These models require more memory than is available on the board, and the computation required in the forward pass is too large. We intend to solve this by using the cross-word models only in the backward direction. Another alternative would be to use the cross-word models to rescore all of the N-Best hypotheses, which could be done relatively quickly. In any case, we decided to make the system work with cross-word models only after we had achieved real time with simpler non-cross-word models.

As we can see, the results using the WP grammar are quite good. Even without the cross-word models, we find the correct sentence 97.6% of the time within the first 20 choices and 98% of the time within the first 100 choices. When we use a beamwidth that gives us real time, we see only a very slight degradation in accuracy. However, as we stated earlier in this paper, the WP grammar is unrealistically easy, both in terms of recognition accuracy and computation. We show these results only for comparison with other real-time recognition results on the RM corpus.

Recognition with the class grammar is much harder due to higher perplexity and the fact that all words are possible at any time. The word error with cross-word models is 4.7%. For the N-Best conditions with the CG grammar we note a larger difference between the sentence errors at 20 and 100 choices. In contrast to the WP grammar in which there are a limited number of possibilities that can match well, here more sequences are plausible. We give the N-Best results for three different speed conditions. The first has a very conservative beamwidth. The second runs at 1.2 times real-time, and the third runs faster than real time. We can see that there is a significant degradation due to pruning errors when we force the system to run in real time.

There are several approaches that are available to speed up the forward pass considerably. Since the forward pass is used for pruning, it is not essential that we achieve the highest accuracy. In those rare cases where the N-Best finds a different top choice sentence than the forward pass, and this new top choice also is accepted by natural language, we will simply have a delay equal to the time taken for the N-Best backward search. The most promising method for speeding up the forward search is to use a phonetic tree in which the common word beginnings are shared. Since most of the words are pruned out after one or two phonemes, much of the computation is eliminated.

Conclusion

We have achieved real-time recognition of the N-Best sentences on a commercially available board. When we use a WP grammar, there is no loss in accuracy due to real-time limitations. However, currently, when using a class grammar there is a degradation. We expect this degradation to be reduced as planned algorithm improvements are implemented.

Most of the increase in speed came from algorithm modifications rather than from fast hardware or low-level coding enhancements, although the latter improvements were substantial and necessary. All the code is written in C so there is no machine dependence. All told we sped up the N-Best computations by a factor of 20,000 with a combination of algorithms, code optimization, and faster hardware.

Acknowledgement

This work was supported by the Defense Advanced Research Projects Agency and monitored by the Office of Naval Research under Contract No. N00014-89-C-0008.

References

[1] Bisiani, R., "Plans for PLUS hardware". *Proceedings*

of the DARPA Speech and Natural Language Workshop Cape Cod, October 1989 (1989).

[2] Chow, Y-L. and Schwartz, R.M., "The N-Best Algorithm: An Efficient Procedure for Finding Top N Sentence Hypotheses". *Proceedings of the DARPA Speech and Natural Language Workshop* Cape Cod, October 1989.

[3] Derr, A., and Schwartz, R.M., "A Simple Statistical Class Grammar for Measuring Speech Recognition Performance". *Proceedings of the DARPA Speech and Natural Language Workshop* Cape Cod, October 1989.

[4] Lowerre, B., "The Harpy Speech Recognition System", *Doctoral Thesis* CMU 1977.

[5] Murveit, H., "Plans for VLSI HMM Accelerator". *Proceedings of the DARPA Speech and Natural Language Workshop* Cape Cod, October 1989.

[6] Rohlicek, J.A., Chow, Y-L., and Roucos, S., "Statistical Language Modeling Using a Small Corpus from an Application Domain". *Proceedings of the DARPA Speech and Natural Language Workshop* Cambridge, October 1987. Also in *Proceedings of the ICASSP 88*, pp. 267-270, April, 1988.

[7] Schwartz, R.M., Chow, Y., Kimball, O., Roucos, S., Krasner, M., and Makhoul, J. Context-Dependent Modeling for Acoustic-Phonetic Recognition of Continuous Speech". *Proceedings of the ICASSP 85*, pp. 1205-1208, March, 1985.

[8] Schwartz, R.M., and Austin, S.A., "Efficient, High-Performance Algorithms for N-Best Search". *Proceedings of the DARPA Speech and Natural Language Workshop* Hidden Valley, June 1990.