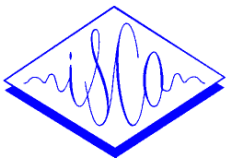

2nd International Conference on
Spoken Language Processing (ICSLP 92)
Banff, Alberta, Canada
October 12-16, 1992

SPEECH UNDERSTANDING ON A MASSIVELY PARALLEL COMPUTER *

Sang-Hwa Chung and Dan Moldovan

Department of Electrical Engineering-Systems
University of Southern California, Los Angeles, California 90089-2562

10.21437/ICSLP.1992-363

ABSTRACT

In this paper, we present a parallel computational model for the integration of speech and natural language processing (NLP). We have developed a parallel speech understanding system on the semantic network array processor (SNAP), a massively parallel computer developed at the University of Southern California. The parallel speech understanding algorithm is based on a memory-based parsing scheme. The key to the integration of speech and linguistic processing is the construction of a hierarchically structured knowledge base. The processing is carried out by passing markers parallelly through the knowledge base. Speech-specific problems like insertion, deletion, substitution, word boundary problems, and multiple hypotheses problems were analyzed and their parallel solutions were provided. The experimental results show that the processing time increases linearly with the length of target sentences, and increases *logarithmically* with the size of the knowledge base. This demonstrates that a massively parallel approach provides a viable platform for integrating speech and NLP on larger domains.

1. INTRODUCTION

The integration of speech and NLP became a key issue in spoken language processing area because speech recognition techniques at the word-level without higher level knowledge was not good enough.

Error rates are fairly high even for the best currently available systems. The CMU's SPHINX system [4] is considered to be one of the best speaker-independent continuous-speech recognition systems today. But even the SPHINX system has a word recognition accuracy of only 70.6% for speaker-independent, continuous-speech recognition with a vocabulary of 1000 words, when recognizing individual words in sentences without using syntax or semantic information[4].

Clearly, we need some forms of higher level knowledge to better understand speech. The integration of speech and NLP helps to improve performance by resolving multiple ambiguous hypotheses using the information provided by the syntactic, semantic and discourse knowledge sources.

However, the integration requires high volume of computations through various levels of knowledge sources. Therefore, an integrated system implemented on uniprocessor environment will face a scalability problem as the system increases the knowledge base size for more complex and broader domains. The processing time of an integrated system on uniprocessors is proportional to the size of the knowledge base. As the size of the knowledge base grows, the speed performance may become unacceptably low.

The high volume of computational requirement of the speech understanding algorithms calls for new approaches to parallel hardware design. The availability of massively parallel computers with their speed advantage brings new hope for real-time speech understanding on larger domains.

In this paper, we present a parallel computational model for the integration of speech and NLP. We have developed a parallel speech understanding system on the semantic network array processor (SNAP), a massively parallel computer developed at USC[5]. SNAP is designed for semantic network processing with a reasoning mechanism based on marker-passing. The SNAP-1 prototype[3] designed in our laboratory consists of a processor array and an array controller. The processor array stores the nodes and links of a semantic network, and has the capability to modify the semantic network by propagating markers. The processor array consists of 160 Texas Instruments TMS320C30 DSP microprocessors. The 160 processor array is organized into 32 tightly-coupled clusters of 5 processors each. Each cluster manages 1024 nodes and has access to 32K nodes of semantic network.

We adopted the memory-based parsing and parallel marker passing schemes as the underlying philosophy to build the system. Memory-based parsing is a rather new approach to natural language understanding compatible with the availability of massively parallel computer systems. Parsing is viewed as a memory search process which identifies patterns in the memory network, and provides interpretation based on the syntactic and semantic relations between patterns.

There are a number of very specific problems in speech input which differ from typed input such as insertion, deletion, substitution problems, word boundary problem, and multiple hypotheses problem. In a parallel processing environment, such problems become even more complicated. We analyze the speech specific problems in detail, and show their solutions on the SNAP system.

2. INTEGRATION OF SPEECH AND NLP ON SNAP

The Phonetic Engine (PE)[1] by Speech Systems Incorporated (SSI) is used as the speech processing (SP) module (or the speech input front-end) to provide a phonetic codes input to SNAP. The PE can handle speaker-independent continuous speech input with a large vocabulary of up to 40,000 words in real-time.

The knowledge base for the speech understanding system is composed of concept sequences and phoneme sequences organized hierarchically. A *concept sequence* is a basic building block of sentences in the memory-based parsing. In each concept sequence, concept nodes are connected by *first, next* and *last* links. Similarly, in each *phoneme sequence*, phonemes are connected by *pfirst, pnext* and *plast* links. Concept sequences are hierarchically stored in the memory network. In other words, more general concept sequences are placed on the higher level, and more specific concept sequences are

*This research has been funded by National Science Foundation Grant No. MIP-9009109.

placed on the lower level. This hierarchy is called a *concept sequence hierarchy.* Phoneme sequences exist at the lowest level of the concept sequence hierarchy attached to the corresponding concept nodes.

Our system works in the Air Traffic Control(ATC) domain developed by SSI for training air traffic controllers. `Tiger six sixteen, reduce speed to one five zero` is a typical target sentence in the ATC domain. Input phonetic codes generated by the PE is represented by a sequence of integers in which each integer corresponds to a phonetic code. The ATC domain contains a vocabulary of approximately 200 words and a network of approximately 1400 semantic nodes.

2.1 The Alignment Scoring Model

One of the early tasks to perform in speech understanding is to find the best matches of an input sequence of phonetic codes with phoneme sequences. To evaluate the matches, we use a codebook derived during the development process using automatically labeled speech data. The codes represent acoustic events having some ambiguity with respect to phonemes. That is, two or more successive phonemes may be time-aligned with a single code and two or more successive codes may be time-aligned with a single phoneme.

To compute an alignment score between the phonetic code sequence S and the sentence transcription T, we first assign score values to the time-aligned subsequences of S and T, and then take the sum of these score values. To compute the score of the time-aligned subsequences, we account for: 1) each alignment between a code and a phoneme, 2) the number of successive phonemes aligned with the same code, and 3) the number of successive codes aligned with the same phoneme. To express the score of an alignment, we introduce three matrices:

- *X(code, phoneme)* - each element x_{ij} is the score to align *code* i with *phoneme* j. The X matrix is generally known as a *confusion matrix.*
- *Y(code, #phoneme)* - each element y_{ij} is the score to align *code* i with number(j) of successive phonemes.
- *Z(phoneme, #code)* - each element z_{ij} is the score to align *phoneme* i with number(j) of successive codes.

The score of an entire utterance is simply the sum of the scores of the time-aligned subsequences of the entire utterance. Some of the speech-specific problems like insertion, deletion and substitution are handled nicely by the alignment scoring model described above[2].

2.2 The Functional Structure of the System

The architecture of the integrated memory-based speech understanding system is shown in Figure 1. The speech understanding (SU) module performs: phoneme prediction, phoneme activation, word boundary control, oversegmentation control, and undersegmentation control. The natural language understanding (NLU) module performs: word prediction, word activation, multiple hypotheses control, and sentence generation. The knowledge base containing concept sequences and phoneme sequences is stored in the SNAP array, and the scoring information including X, Y and Z matrices is in the SNAP controller.

In principle, the parallel speech understanding algorithm is based on a combination of top-down prediction and bottom-up activation in the concept sequence hierarchy. In other words, top-down prediction decides the candidates to be evaluated next and bottom-up recognition activates a set of phonemes from a given phonetic code.

As shown in Figure 1, a circular path exists between the NLU module and the SU module as follows: word prediction - phoneme prediction - phoneme activation - word boundary

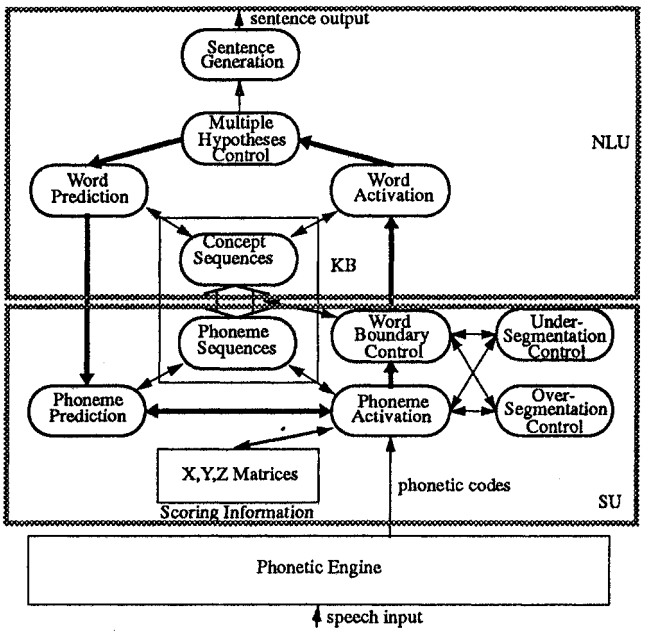

Figure 1: The functional structure of the parallel speech understanding system.

control - word activation - multiple hypotheses control - word prediction. The operation starts in the NLU by predicting the first words in all concept sequences. This in turn impacts the prediction of the first phonemes of these predicted words. Next, the system accepts a phonetic code as speech input, and via X, Y, Z matrices all the relevant phonemes are activated. The candidates of predicted and activated phonemes trigger further phoneme predictions. This process repeats until some word hypotheses are formed. This coincides with the activation of some words, and the process is moved to the NLU module. Here, through a similar process with the one at the SU level, only the coincidence of predicted and activated words triggers further word predictions. All words originally predicted, but not activated, receive cancellation markers. Then, the cycle repeats.

The repeated top-down prediction and bottom-up activation are performed on this path until all phonetic code inputs are processed. The actual predictions and activations are performed by propagating markers parallelly in the knowledge base. The segmentation problems are handled by the oversegmentation and undersegmentation control.

The processing of speech input on the parallel speech understanding system requires the creation and movement of markers on the memory network. Some of markers performing important functions are:

- *P-Markers* indicate the next possible nodes (or phonemes) to be activated in the concept sequence (or phoneme sequence). They are initially placed on the first nodes of concept sequences and phoneme sequences, and move through the *next* (or *pnext*) link.
- *A-Markers* indicate activations of nodes. They propagate upward through the concept sequence hierarchy.
- *I-Markers* indicate instantiations of activated nodes. Activated concept nodes are finally identified by I-Markers.
- *C-Markers* indicate cancellations of activated nodes. Because of multiple hypotheses, some I-Markers may be cancelled or invalidated later as their scores become inferior.

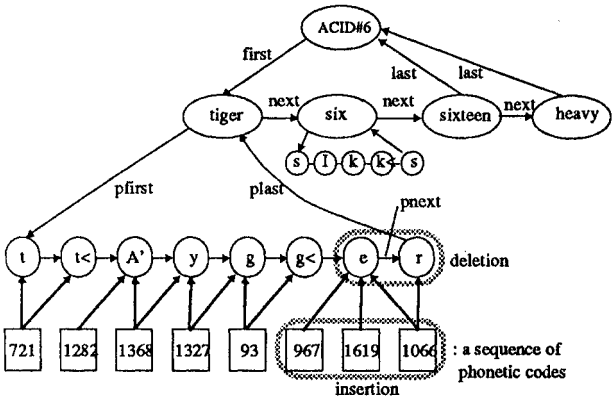

Figure 2: An example of handling the insertion and deletion problems.

2.3 Marker-passing Solutions for Insertion and Deletion Problems

During the phoneme sequence recognition process, the insertion and deletion problems occur in almost all words. It is critical to handle these problems properly for a successful recognition of phonemes. The alignment scoring model provides the necessary information to solve the problems by the X,Y,Z matrices.

An example describing these problems is shown in Figure 2, where only a part of the concept sequence hierarchy for the ATC domain is depicted. In this figure, we show the time alignment between the subsequence of phonetic codes: 721 1282 1368 1327 93 967 1619 1066, and the phoneme sequence for the word *tiger*: t t< A' y g g< e r.

As an example, let us assume that we have processed the subsequence of codes: 721 1282 1368 1327 93. The remaining codes can be handled as follows (dual prediction markers, P(-1) and P(0), are used to keep the previously used P-Marker as well as the current P-Marker):

1. Code *967* is accepted now from the PE. By consulting the X matrix in the controller, phoneme *e* is activated. The scoring process is as follows: Score(P(0)) = Previous_Score(P(0)) + Score(A), where Score(A) $= X(967, e) + Y(967, 1) + Z(e, 1)$. Then, P(0) is propagated to phoneme *r* from phoneme *e* through the *pnext* link. Phoneme *e* also keeps P(-1) to prepare against a possible insertion.
2. When code *1619* arrives, phoneme *e* is activated again, instead of phoneme *r*. That is, an insertion exists. The insertion handling routine calculates the score of the A-Marker: Score(A) $= X(1619, e) + Y(1619, 1) + Z(e, 2)$ where the previous $Z(e, 1)$ need not be cancelled because scores in the Y, Z matrices only contain offset values to avoid unnecessary computations. After adding the score of A to P(-1), a new P(0) is propagated again to phoneme *r*.
3. When code *1066* arrives, phoneme *e* and phoneme *r* are activated together. That is, both an insertion and a deletion occur at the same time. By this, we can assume that there exist no more insertions to phoneme *e*. Now, the score of the A-Marker is calculated as: Score(A) $= X(1066, e) + Y(1066, 1) + Z(e, 3)$. Again, a new P(0) adding the score of A is propagated to phoneme *r*. Here, a collision between P(0) and A exists, and the scoring process for phoneme *r* begins. Because phoneme *r* is the last phoneme activated in the phoneme sequence, an A-Marker propagates upward through the concept sequence hierarchy, and finally a new P(0) arrives at the first phoneme of the phoneme sequence for concept *six*.

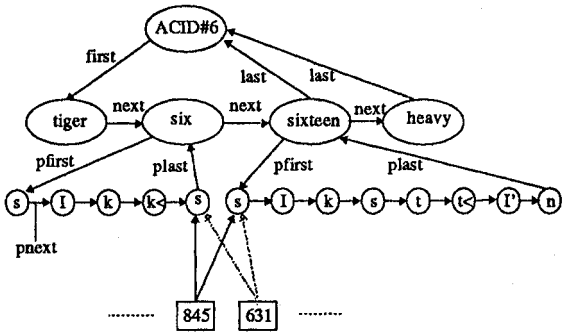

Figure 3: An example of the word boundary problem.

In the example above, we have explained how to handle the insertion and deletion problems. Because multiple hypotheses are evaluated at the same time, similar recognition processes with the one described above are performed in parallel throughout the memory network.

2.4 Marker-passing Solution for the Word Boundary Problem

The *word boundary problem* occurs when the last phoneme of a phoneme sequence is activated and the first phoneme of the next phoneme sequence is predicted. Basically, we control the word boundary problem by the insertion and deletion handling algorithms with the aid of the *concept sequence hierarchy*. An example describing the coarticulated phrase **six sixteen** is shown in Figure 3. In this example, the last phoneme of concept **six** and the first phoneme of concept **sixteen** are located on the word boundary and represented by the same phoneme type *s*. Let us assume that dual prediction markers P(-1) and P(0) are located on the last two phonemes of concept **six** respectively.

As shown in the figure, when code *845* is evaluated, the two phonemes on the boundary are activated together indicating a deletion problem across the boundary. Because both phonemes have P&A collisions, dual prediction markers are advanced twice reflecting newly calculated scores. As a result, P(-1) and P(0) are located on the first and second phonemes of concept **sixteen**. As shown in the figure, the next input code *631* activates those two phonemes on the boundary again. However, only the first phoneme of concept **sixteen** gets a P&A collision indicating that the word boundary problem is resolved and the alignment process for concept **sixteen** begins.

2.5 Multiple Hypotheses Resolution

Multiple hypotheses problems can not be completely resolved with the information available at the phoneme sequence level. The SU module activates multiple competing words, or even the same words repeatedly when insertion problems exist. When A-Markers are propagated up through the concept sequence hierarchy at a word boundary, several candidates may exist at any merging point.

A merging point exists when a concept node contains multiple incoming *last* links, *isa* links, or *next* links. A merging point also exists when a concept node has multiple *plast* links coming from multiple phoneme sequences representing various pronunciations of the concept node.

When multiple candidates arrive at a merging point, the concept node in the merging point might have been either

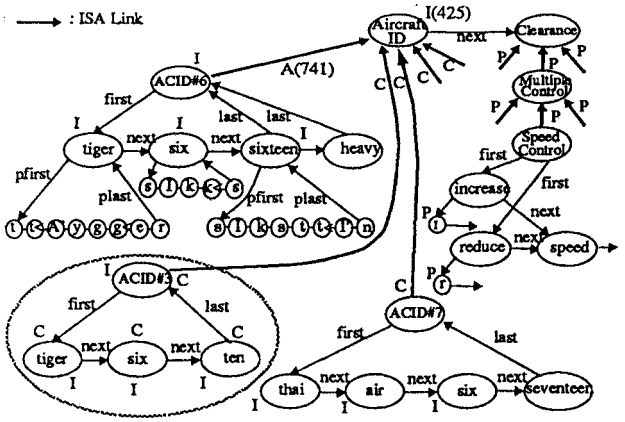

Figure 4: An example of multiple hypotheses resolution.

already activated from the evaluations of the previous input codes or first visited this time. To get the best candidate, the scores carried by the candidates' A-Markers are compared. If the concept node was activated previously, the score of the I-Marker is also compared with the scores of the A-Markers. The candidates with lower scores are cancelled by propagating C-Markers down through the concept sequence hierarchy. As a result, the concept node gets a new I-Marker containing the highest score.

An example for the multiple hypotheses resolution is shown in Figure 4. In this figure, the concept node *Aircraft ID* was previously activated by the hypothesis: `tiger six ten`, and the score of the I-Marker for the node is 425. Although `tiger six ten` is apparently different from `tiger six sixteen`, it is still possible to activate this hypothesis with a low score.

When a new hypothesis: `tiger six sixteen` arrives at the node *Aircraft ID* with the score of the A-Marker, 741, the previous hypothesis is rejected because of its lower score. C-Markers are parallelly propagated down through all possible links in the concept sequence hierarchy except the link to the newly selected hypothesis. When C-Markers collide with I-Markers during the propagation, the I-Markers are cancelled. To protect partially evaluated hypotheses, a C-Marker in each node stops its propagation when the node does not contain an I-Marker. For example, `thai air six seventeen` is still in the middle of evaluation, and later, can be the hypothesis with the highest score. Thus, the I-Markers on the nodes *thai*, *air* and *six* should be kept.

After resolving the multiple hypotheses problem, a P-Marker is propagated to the concept node *Clearance* through the *next* link. From the node *Clearance*, P-Markers are propagated down to the first phonemes of the next possible phoneme sequences as shown in the figure. The activation, cancellation, and prediction operations are performed simultaneously for all possible hypotheses by marker passing.

3. EXPERIMENTS AND CONCLUSIONS

For the experiments, we used the ATC domain with 1400 semantic network nodes. The experiments were performed on the SNAP-1 prototype with 16 clusters. (Currently 16 clusters are available out of 32 clusters.) We used 60 different continuously-uttered sentences. Each sentence was spoken by four different speakers. Each sentence is on the average 10 words long, or 50 phonemes long. Currently, the parallel speech understanding system understands about

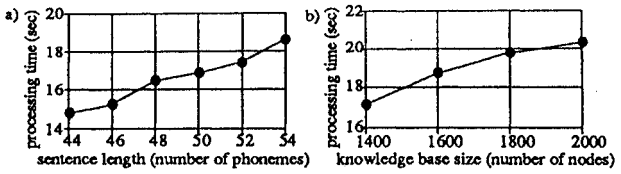

Figure 5: a) Processing time vs. target sentence length on 16 clusters, b) Processing time vs. knowledge base size on 16 clusters.

80% of sentences correctly. The processing of one sentence takes 17 seconds on the average, approximately 1.7 seconds per word. Better performance is expected when the full 32 clusters system becomes available.

When a target sentence is processed in the memory network, the length of the critical path to parse the target sentence is roughly determined by the length of phoneme sequences representing the sentence and the depth of the concept sequence hierarchy. The effect of target sentence length on processing time is shown in Figure 5.a. The sentence length is measured by the number of phonemes. The processing time increases linearly with the number of phonemes.

To analyze the scale-up effect, we increased the size of the knowledge base by adding more concept sequences and phoneme sequences to the concept sequence hierarchy, and measured the variation of processing times as function of the knowledge base size. As shown in Figure 5.b, the processing time increases *logarithmically* with the size of the knowledge base. This experimental result is promising since realistic applications require very large knowledge bases, which may be prohibitive for many sequential machines to handle. More experiments with larger knowledge bases are necessary to get more accurate measurements of the relationship between the knowledge base size and the processing time. We are now installing the Radiology domain provided by the SSI consisting of a vocabulary of approximately 2K words and a network of approximately 10K semantic nodes, and are in the process of testing the system performance on this larger knowledge base.

REFERENCES

[1] M.T. Anikst, W.S. Meisel, M.C. Soares, and K.F. Lee. "Experiments with Tree-Structured MMI Encoders on the RM Task," *Proceedings of Speech and Natural Language Workshop*, June 1990.

[2] M.T. Anikst and D.J. Trawick. "Training Continuous Speech Linguistic Decoding Parameters as a Single-Layer Perceptron," *Proceedings of International Joint Conference on Neural Networks*, vol. 2, pp. 237-240, 1990.

[3] R. DeMara and D.I. Moldovan. "The SNAP-1 Parallel AI Prototype," *Proceedings of the 18th Annual International Symposium on Computer Architecture*, 1991.

[4] K.F. Lee. "Large-Vocabulary Speaker-Independent Continuous Speech Recognition: The Sphinx System," *Technical Report* CMU-CS-88-148, Department of Computer Science, Carnegie-Mellon University, 1988.

[5] D.I. Moldovan, W. Lee, C. Lin, and S.H. Chung. "Parallel Knowledge Processing on SNAP," *Proceedings of ICPP-90*, vol. I, pp. 474-481, 1990.