PASS: A Parallel Speech Understanding System

Sang-Hwa Chung

Abstract

A key issue in spoken language processing has become the integration of speech understanding and natural language processing (NLP). This paper presents a parallel computational model for the integration of speech and NLP. The model adopts a hierarchically-structured knowledge base and memory-based parsing techniques. Processing is carried out by passing multiple markers in parallel through the knowledge base. Speech-specific problems such as insertion, deletion, and substitution have been analyzed and their parallel solutions are provided. The complete system has been implemented on the Semantic Network Array Processor(SNAP) and is operational. Results show an 80% sentence recognition rate for the Air Traffic Control domain. Moreover, a 15-fold speed-up can be obtained over an identical sequential implementation with an increasing speed advantage as the size of the knowledge base grows.

I. Introduction

Despite several decades of research activity, spoken language processing still remains a difficult field. The ultimate goal of speech research is to create an intelligent assistant, who listens to what a user tells it and then carries out the instructions. An apparently simpler goal is the listening typewriter, a device which merely transcribes whatever it hears with only a few seconds delay. The listening typewriter seems simple, but in reality the process of transcription requires almost complete understanding as well. Today, we are still quite far from these ultimate goals, but progress is being made.

A key issue in spoken language processing has become the integration of speech understanding and natural language processing (NLP). Their effective integration offers higher potential recognition rates than recognition using word-level knowledge alone. Without higher level knowledge, error rates for even the best currently available systems are fairly high. For example, CMU's Sphinx system [6] 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 [6].

Clearly, we need some forms of high-level knowledge to better understand continuous speech. The integration of speech and NLP resolves multiple ambiguous hypotheses using syntactic, semantic and contextual knowledge sources. Since this requires sizable computation involving multiple levels of knowledge sources, speed performance degrades on realistic knowledge bases suitable for broader, complex domains.

The high volume of computational requirement of integrated speech understanding algorithms calls for new approaches to parallel processing. Recent work on parallel parsing is relatively limited. Moreover, most of the works relate to written language. Huang and Guthrie [5] proposed a parallel model for natural language parsing based on a combined syntax and semantics approach. Waltz and Pollack [9] investigated parallel approaches under paradigms related to the connectionist model. Giachin and Rullent [4] developed a parallel parser called SYNAPSIS for spoken natural language using a Transputer-based distributed architecture. They used a case frame-based parsing scheme and reported a sentence recognition accuracy of about 80% from continuously-uttered sentences, on average 7 words long, with a dictionary of 1000 words. However, SYNAPSIS was not implemented on a real parallel system. The analysis of one sentence by a sequential version of SYNAPSIS took on the average 40 seconds on a SYMBOLICS.

In this paper, we describe how the scalability problem can be addressed in the integrated PArallel Speech understanding System called *PASS*. A memory-based parsing model and parallel marker-passing schemes form the underlying philosophy of the system. We show how to handle the major speech-specific problems such as insertion, deletion, and substitution using tightly-coupled iterations between low-level phoneme sequences and higher-level concepts. Beyond simply recognizing speech

Manuscript received February 22, 1995; accepted June 23, 1995.

The anthor is with Department of Computer Engineering, Pusan National Univ., Pusan, Korea.

and converting it into text, PASS employs the underlying meaning representation through parallel speech understanding. We describe an operational implementation of our integrated approach and analyze its performance on a real parallel machine.

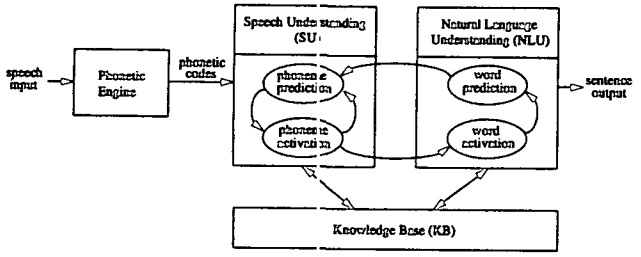


Fig. 1. The PASS environment.

II. Integrated Speech and Natural Language Processing

The objective of PASS is to provide a parallel system which integrates speech and high-level NLP to improve processing speed and tractable domain size.

1. System Overview

As shown in Fig. 1, PASS contains the natural language understanding (NLU) module, the speech understanding (SU) module and the knowledge base. The inputs to PASS are provided by the Phonetic Engine [7] manufactured by Speech Systems Incorporated (SSI). The Phonetic Engine provides a stream of input phonetic codes for processing. It performs signal processing on speaker-independent continuous speech in real-time.

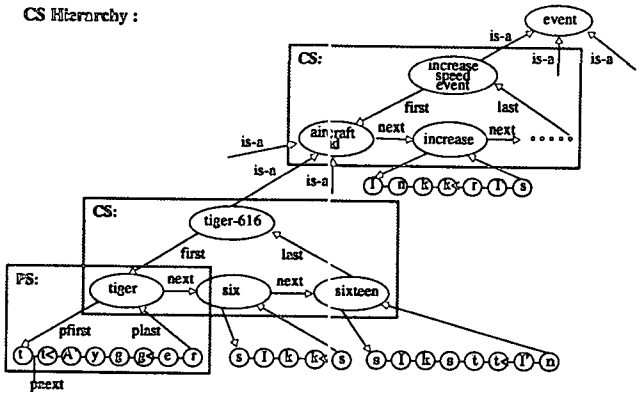


Fig. 2. Hierarchical knowledge base including phoneme sequence level - ATC domain.

The input codes provided by the Phonetic Engine are evaluated in the SU module to find the matching phoneme sequences. The predictions provided by the NLU module allow the SU module to handle multiple hypotheses efficiently. The NLU module guides the scope of the search space. Word candidates activated by the SU module are further evaluated in the NLU module to construct meaning representations and generate a sentence output. The predictions and activations are performed in parallel by markers throughout the knowledge base.

2. Hierarchical Knowledge Base

We use hierarchically organized knowledge bases to support close interaction between several levels of knowledge sources. A *concept sequence* (CS) is a basic building block in memory-based parsing. Each CS represents the underlying meaning of a possible phrase or sentence within a domain. In each CS, concept sequence element (CSE) nodes are connected by *first*, *next* and *last* links. Similarly, in each *phoneme sequence* (PS), phoneme sequence element (PSE) nodes are connected by *p-first*, *p-next* and *p-last* links to form words. More general CSs are placed at higher levels, and more specific CSs are placed at lower levels. This type of memory network is called a *CS hierarchy*. Phoneme sequences, which are attached to the corresponding concept nodes, reside at the lowest level of the concept sequence hierarchy.

Fig. 2 shows a part of a hierarchical knowledge base from the Air Traffic Control (ATC) domain developed for training air traffic controllers [8]. We have adapted the ATC domain to support a vocabulary of approximately 200 words using a hierarchical semantic network of approximately 1400 nodes. In Fig. 2, tiger-616 is a CS root node and tiger, six and sixteen are the corresponding CS element nodes. Phoneme sequences are attached to these CS element nodes as shown in Fig. 2.

A layered structure makes it possible to process knowledge from the phonetic level to the contextual level by representing knowledge using a layered memory network. After phonemes are processed and word hypotheses are formed, linguistic analysis can be performed based on the syntactic, semantic and contextual constraints embedded in the knowledge base.

3. The Alignment Scoring Model

It is difficult to correctly align input phonetic codes with target phoneme sequences because the phonetic codes contain the segmentation problems such as insertion, deletion and substitution. The code/phoneme statistics collected by SSI provide the necessary information for the alignment process.

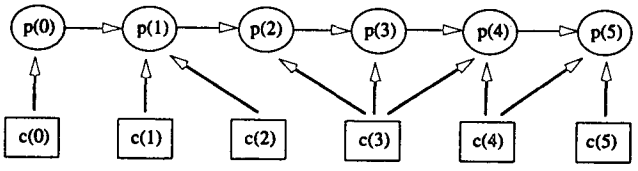


Fig. 3. A possible alignment between input codes and a target transcription.

An initial task to be performed in speech understanding is finding the best correspondence of input codes to phoneme sequences. To evaluate each match, a codebook is used, which was derived from automatically labelled speech data, collected from several speakers. 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. Our system accepts 1644 different phonetic codes generated by the Phonetic Engine, which map to 49 phonemes. Each input code is assigned by an integer between 0 and 1643.

The above can be described in terms of an alignment scoring model[1]. A sequence S consists of separate input codes $c(i)$ and is denoted by $S = \{ c(i) : i=0,\ldots,N \}$. To find the sentence which produced S, the memory network is searched for a sentence transcription $T=\{ p(j): j=0,\ldots,M \}$ consisting of phonemes, each labeled $p(j)$. The correspondence of S to T which maximizes the alignment score is chosen as output.

A subsequence of codes $\{ c(i) : i = i_0,\ldots,i_n\}, i_0 \le i_n$, can be time-aligned with a single phoneme $p(j)$. Conversely, a subsequence of phonemes $\{ p(j) : j = j_0,\ldots, j_n\}$, $j_0 \le j_n$, can be time aligned with a single code, $c(i)$. A possible alignment is illustrated in Fig. 3. Here $c(0)$ is aligned with $p(0)$; $p(1)$ is aligned with $\{ c(1), c(2)\}$; $c(3)$ is aligned with $\{ p(2), p(3), p(4)\}$; $p(4)$ (the last phoneme of the previous subsequence) is also aligned with $c(4)$; $c(4)$ (the last code of the previous subsequence) is also aligned with $p(5)$; and finally, $p(5)$ is also aligned with $c(5)$.

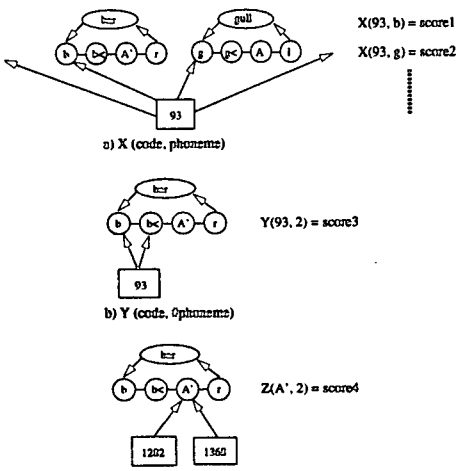


Fig. 4. An example of applying X, Y and Z matrices.

4. X, Y, and Z Matrices

To compute the alignment score between S and T, score values are computed for the time-aligned subsequences of S and T. The model accounts for: 1) each alignment betweena 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, three matrices are required:

- *X(code, phoneme)* - each element x_{ij} is a 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 a score to align *code* i with number(j) of successive phonemes.
- *Z(phoneme, #code)* - each element z_{ij} is a score to align *phoneme* i with number(j) of successive codes.

Fig. 4 shows an example of applying the X, Y and Z matrices. In Fig. 4-a, the phonetic code *93* is aligned with the first phonemes of PSs: bar & gull, and other phonemes in the knowledge base with different X matrix scores. Alignment examples of applying the Y and Z matrices are shown in Fig. 4-b and c, respectively.

For each input code, alignment scores are calculated by consulting the X, Y and Z matrices, and the score of an entire utterance is the sum of the scores of the time-aligned subsequences in the utterance. For example, the score for Fig. 3 is computed as:

$$\begin{aligned}\text{Score} = &\{ X(c(0),p(0)) + Y(c(0),1) + Z(p(0),1) \} \\ &+ \{X(c(1),p(1)) + X(c(2),p(1)) + Y(c(1),1) \\ &+ Y(c(2),1) + Z(p(1),2)\} + \{ X(c(3),p(2)) \\ &+ X(c(3),p(3)) + X(c(3),p(4)) + Y(c(3),3) \\ &+ Z(p(2),1) + Z(p(3),1) + Z(p(4),2) \} \\ &+ \{ X(c(4),p(4)) + X(c(4),p(5) + Y(c(4),2) \\ &+ Z(p(5),2)\} + \{X(c(5),p(5)) + Y(c(5),1)\}\end{aligned}$$

Each set of brackets indicates a sub-group of possible alignments. For instance, the first set of brackets indicates an alignment between $c(0)$ and $p(0)$.

The individual scores for the X, Y and Z matrices are logarithms of probabilities and/or probability ratios [1]. They are scaled (by choosing an appropriate base of the logarithm) into the range of an 8-bit integer between -128 and 127. The scores for Y and Z matrices are offset with the single alignment scores, such as $Y(c(i),1)$ and $Z(p(j),1)$, to avoid unnecessary computations during the scoring process.

Although we have shown the final alignment scores in the above example, actual computations include some intermediate scores. For example, in Fig. 3, suppose we have already processed the phonetic codes $c(0)$ and $c(1)$, and $p(1)$ aligned with a single code with the score of $Z(p(1),1)$. When the next code, $c(2)$, is accepted, a new alignment is formed with the score of $Z(p(1),2)$. In this case, to extend the alignment matching a single code to 2 successive codes, we only need to add $Z(p(1),2)$ instead of also subtracting $Z(p(1),1)$. This is possible because the individual scores for Y and Z matrices are already offset.

Insertion, deletion and substitution can be handled in terms of the alignment scoring model. Specifically, insertion problems are

handled by the Z matrix; deletion problems are handled by the Y matrix; substitution problems are handled by the X matrix.

5. The Functional Structure of PASS

The architecture of the integrated memory-based speech understanding system is shown in Fig. 5. Dark arrows indicate execution flow and light arrows show information flow. The SU module performs: *phoneme prediction, phoneme activation, word boundary detection, insertion control,* and *deletion control.* The NLU module performs: *word prediction, word activation, multiple hypotheses resolution, meaning representation construction,* and *sentence generation.* The knowledge base containing concept sequences and phoneme sequences is distributed to each parallel processing element and the scoring information including X, Y and Z matrices is in a host or central controller.

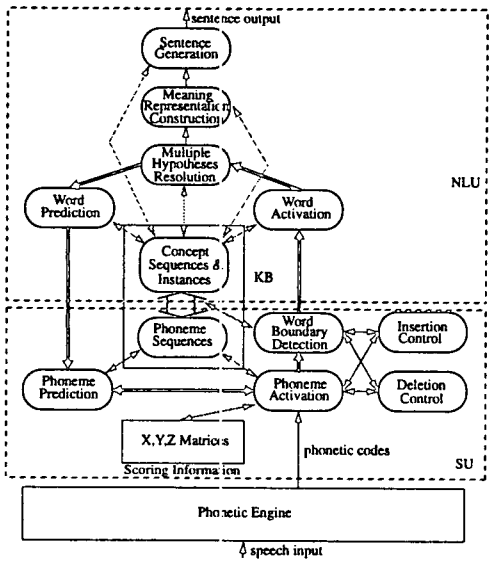


Fig. 5. Modules within PASS.

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 Fig. 5, a circular path exists between the NLU module and the SU module as follows: word prediction → phoneme prediction → phoneme activation → word boundary control → word activation → multiple hypotheses control → word prediction. The operation starts in the NLU module by predicting the first words in all concept sequences.

This in turn impacts the prediction of the first phonemes for 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 new word hypotheses are formed. This coincides with the activation of corresponding words, and the process is moved to the NLU module. Here, through a process similar to the one at the SU level, only the coincidence of predicted and activated words triggers further word predictions. When a word gets both prediction and activation, a concept instance is generated to construct a meaning representation. All words originally predicted, but not activated, receive cancellation markers. Afterwards, this cycle repeats.

The repeated top-down prediction and bottom-up activation are performed on this circular path until all phonetic code inputs are processed. Once a whole CS is recognized, a CS instance (CSI) is generated as an interpretation of the speech input, and stored in the knowledge base. As shown in Fig. 6, a sequence of input codes are matched against the CS: increase-speed-event in the CS memory using the above algorithm, and the CS instance (CSI): increase-speed-event#3 is dynamically generated in the CSI memory as a result of parsing. The CS and CSI are connected by a CS instance link. From the CSI, the sentence output: "Tiger six sixteen, increase speed to one five zero knots" is generated, as shown in Fig. 6.

The actual predictions and activations are performed by propagating markers in parallel through the knowledge base. The processing of speech input on PASS requires the creation and movement of markers on the memory network. The actual implementation required about 20 different types of markers, including both *fat-markers* and simple *bit-markers.* Fat-markers carry scoring information and move around the memory network by marker propagation operations, while bit-markers only represent certain characteristics of nodes and are not propagated. Some fat-markers performing important functions are:

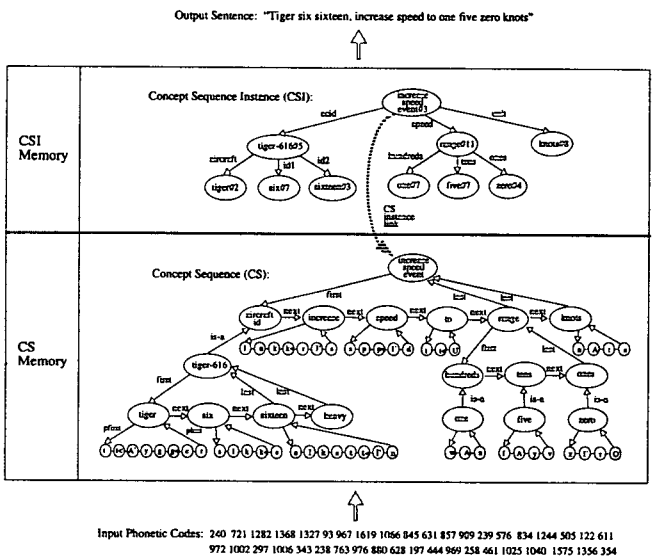


Fig. 6. An example of CSI generated as a result of parsing.

- *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.

6. Marker-passing Solutions for Speech-specific 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 successful recognition of phonemes. The alignment scoring model provides the necessary information to solve the problems by the X,Y, and Z matrices.

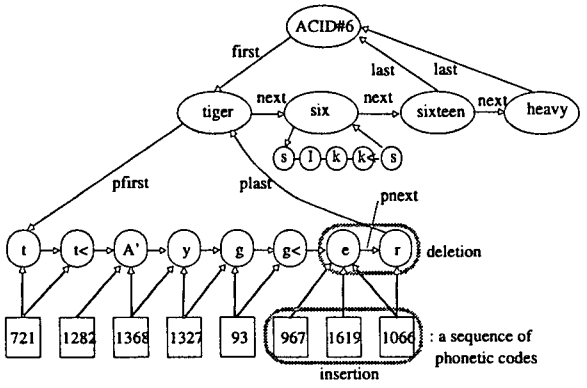


Fig. 7. An Example of Handling the Insertion and Deletion Problems.

However, it is not easy to simultaneously align a sequence of phonetic codes containing such problems with multiple phoneme sequence targets in the memory network, because the multiple targets should be evaluated at the same time, and the next phoneme activations cannot be foreseen while the current phonetic code is being processed. Therefore, when we advance the prediction markers based on the current scoring information, we should consider how to recover from bad expectations in some phoneme sequence candidates, without affecting good expectations in other candidates.

An example describing these problems is shown in Fig. 7, where only a part of the concept sequence hierarchy for the ATC domain is depicted. In this figune, 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 now accepted from the Phonetic Engine. By consulting the X matrix in the central controller, phoneme *e* is activated. The scoring process is as follows: Score(P(0)) = Previous_Score(P(0)) + Score(A), where $\text{Score}(A) = X(967, e) + Y(967, 1) + Z(e, 1)$. 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 theA-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*.

Because multiple hypotheses are evaluated at the same time, recognition processes similar to the one described above are performed in parallel throughout the memory network. Substitution problems are not shown in Fig. 7. When a substitution occurs, the score of the X Matrix for the substituted code-phoneme pair will be low or even below a threshold. Thus, when a substitution problem occurs in a phoneme sequence candidate, the score of the candidate is decreased. This candidate may be rejected when other hypotheses get better scores in any decision point.

7. Multiple Hypotheses Resolution

Multiple hypotheses cannot 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 re-

presenting various pronunciations of the concept node.

In PASS, a scheme for score-based multiple hypotheses resolution is performed in parallel using marker-passing. When multiple candidates arrive at a merging point, the concept node in the merging point might have been either 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.

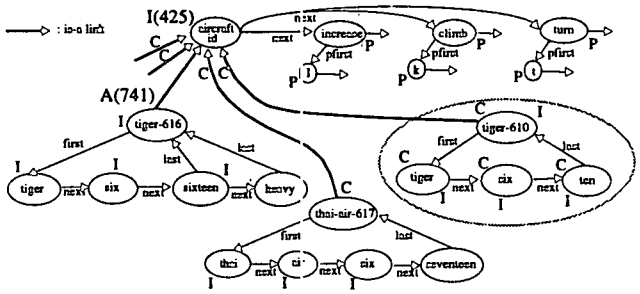


Fig. 8. An Example of Multiple Hypotheses Resolution.

An example for the multiple hypotheses resolution is shown in Fig. 8. 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. Because this hypothesis was not a correct one, the activation score was poor. Although tiger six ten is apparently different from tiger six sixteen, it is still possible to activate this hypothesis with a low score. Basically, any phoneme with the X Matrix score greater than a certain threshold can be activated from an input code regardless of its meaning. So, it is critical to set the threshold properly to prevent unnecessary activations without losing any meaningful information.

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 simultaneously 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 as shown, while later, turning out to be the hypothesis with the highest score. Thus, the I-Markers on the nodes *thai*, *air* and *six* need to be retained.

After resolving multiple hypotheses, P-Markers are propagated to the concept nodes such as *increase*, *climb*, and *turn* through *next* links. From these nodes, P-Markers are further propagated down to the first phonemes of corresponding phoneme sequences. The activation, cancellation, and prediction operations are performed simultaneously for all possible hypotheses.

III. Execution Results

The PASS integrated speech/NLP algorithm has been implemented on the SNAP-1 parallel machine[3] and is operational. Currently, the Air Traffic Control domain is being used and a larger Radiology domain is in development. We describe below the current system configuration and its performance.

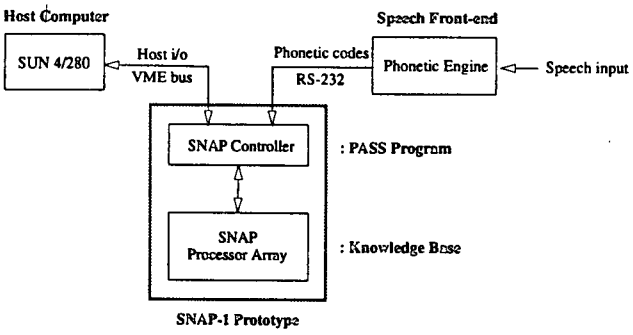


Fig. 9. The PASS system implemented on the SNAP-1 parallel machine.

1. SNAP-1 Parallel Processing Platform

Fig. 9 shows the actual implementation of the PASS system on the SNAP-1 parallel machine. The combined system consists of the SNAP-1 parallel machine, the Phonetic Engine, and a SUN 4/280 (host computer). The speech front-end and the SUN host are connected to the SNAP controller via an RS-232 and a VME bus, respectively.

SNAP-1 is a parallel array processor designed for semantic network processing with a reasoning mechanism based on marker-passing. The SNAP-1 architecture is based on an a multiprocessing array and a dual-processor array controller. The array stores a semantic network of up to 32K nodes and 320K links. The processor array consists of 144 Texas Instruments TMS320C30 DSP microprocessors. The array is organized as 32 tightly-coupled *clusters* of 4 to 5 Processing Elements (PEs) each[1]. Each cluster manages 1024 semantic network nodes.

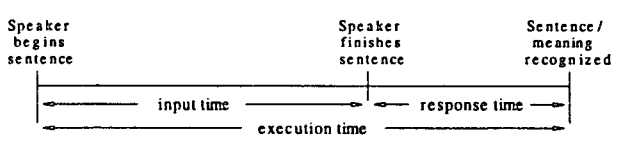


Fig. 10. Definition of execution time and response Time.

1) Presently, 16 clusters are implemented in the full 5 PE configuration while the remaining 16 clusters have 4 PEs each, totaling 144 PEs.

The central controller interfaces the processor array with a SUN4/280 host where application programs are written and compiled. Within the SNAP-C environment, high-level operations and libraries are provided for marker-passing. The parallel operations are initiated through a global bus from the controller which begins each propagation cycle by broadcasting marker instructions to the array. However, most of the work is performed locally though the propagation of markers at nodes within the cluster. Thus, several instructions and multiple propagations can be performed simultaneously.

2. *Recognition Accuracy*

We have analyzed the execution of PASS for the ATC domain on the SNAP-1 platform and a uniprocessor. The essential elements of the domain, including basic concept sequences and phoneme sequences occupy 1,357 semantic network nodes with 5,834 links. Results are reported for a 16 cluster configuration which was the maximum available at the time. The controller clock speed was 32 MHz while the array processors operated at 25 MHz.

We tested 60 different continuously-uttered sentences. Each sentence contained about 11 words and was spoken by four different speakers. An 80% sentence recognition rate was achieved within the domain for these untrained speakers.

3. *Response Time and Scale-up*

We have measured response time as more knowledge is used. The knowledge base size was increased by inserting additional concept and phoneme sequences. For each configuration, results for program running time are reported according to the definitions in Fig. 10. *Execution time* indicates the time elapsed from when the speaker first begins the sentence. Since the speech codes are generated by the Phonetic Engine as the sentence is spoken, PASS begins execution immediately when the first code is generated. This means that the input and processing are overlapped. Thus, the *response time* observed by the user is only the time required to construct meaning representation and generate an output sentence after the last input is received.

Fig. 11 shows response times for various number of semantic network nodes. With a 16-cluster SNAP-1 configuration operating at 25 MHz, response time for the basic ATC domain was 3.7 seconds with input time of about 5 seconds. Thus, near real-time performance can be obtained, while extracting meaning representation and generating a sentence output from untrained continuous-speech input when using a knowledge base of this size. When more nodes were added, response time increased linearly with a small slope. Response time ranged from 3.7 seconds for 1.4K nodes to 23 seconds for 9K nodes.

The identical algorithm is executed on a single TMS320C30 processor at 25 MHz. Response time also increases linearly, but the user must wait over 40 seconds for a response, even when using the basic ATC domain. While both response times increase linearly, a 15-fold speed-up is obtained from the parallel implementation for 9K nodes.

KB Size (nodes)	Response Time (sec)	
	SNAP	Uniprocessor
1400	3.7	41.4
1600	4.3	50.2
1800	4.8	61.4
2200	5.9	75.3
2400	6.2	82.7
2800	7.4	95.6
3200	8.3	108.7
3600	9.4	125.2
4000	10.3	142.7
5000	12.8	158.2
5500	14.0	202.4
6500	16.6	239.3
7500	19.1	273.9
9000	23.0	339.6

Fig. 11. Response time.

The rate at which processing time increases is primarily influenced by the critical path of marker propagation. The critical path is determined by the structure of the knowledge base as it grows. Consider an efficient parallel implementation for a knowledge base which grows *hierarchically*. The critical path corresponds to the maximum depth from the root to the leaves of the hierarchy. Thus performance approaching logarithmic time can be obtained up to the number of processors available.

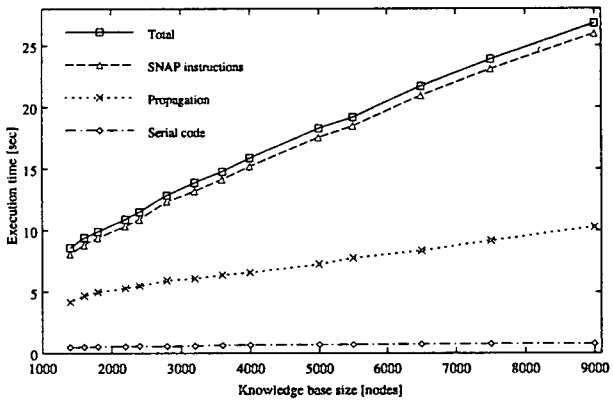


Fig. 12. Execution time.

However, a linguistic knowledge representation typically introduces new nodes at predetermined levels for the concept and phoneme sequences. Therefore, although the knowledge base is organized hierarchically, it maintains a relatively *fixed depth* while its breadth increases. The length of the critical path is roughly fixed, and performance approaching constant time can be obtained in terms of knowledge base size on a parallel machine with sufficient resources. For increases in knowledge base size which are large relative to the processing resources

available, execution time increases at a constant rate so near linear performance will occur on an efficient parallel machine. In PASS, the addition of new nodes does not significantly change the knowledge base depth. Since meaningful increments in knowledge base size exceed the number of processors in the SNAP-1 array, linear performance is obtained.

4. Components of Execution Time

To understand the execution characteristics of the algorithm, we also studied the components of execution time required to process a typical target sentence on SNAP-1 as shown in Fig. 12. The dashed line is for all 26 types of SNAP instructions, including marker-propagation. The dotted line shows that the majority of processing time is spent in the propagation phase. Only a small portion of the code is serial and cannot be executed as SNAP array instructions. The serial portion is about 10% for small knowledge bases and less than 4% percent for larger knowledge bases. Since the reasoning mechanisms are based on marker-propagation, the serial processing time does not depend heavily on the size of the knowledge base.

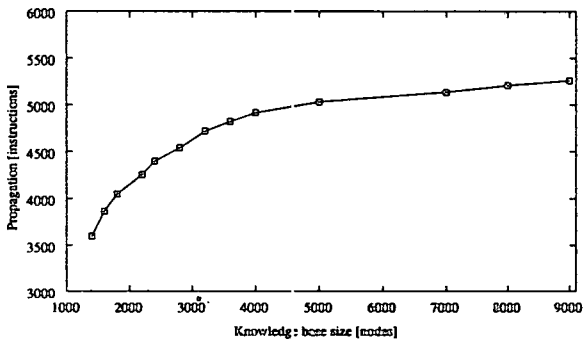


Fig. 13. Increase in number of propagate instructions.

However, as shown in Fig. 13, there is some increase in the total number of propagations required. This occurs because more irrelevant candidates become activated, which must be removed by propagating cancel markers during the multiple hypotheses resolution phase. Although large knowledge bases will add candidates which are not relevant, the number of propagations is not expected to exceed much more than 5000. Most other operations remained relatively constant with processing dominated by *marker set/clear* (12,000 instructions)and *boolean marker operations* (11,000 instructions).

Array Size (clusters)	Response Time (sec)
4	12.2
6	8.2
8	6.3
10	5.2
12	4.4
14	3.8
16	3.7

Fig. 14. Response time vs. array size (KB size: 1.4K).

5. Processor Speed-up

Fig. 14 shows the effect of varying the number of processors while the size of the knowledge base is held constant. Since the capacity of a single cluster is limited to 1024 nodes, the basic ATC domain was used with 4 or more clusters. For each configuration, response time was measured for different sentences and the average processing time was calculated. In general, the performance improves as the number of processors is increased. The improvement levels off when a large number of clusters are used. This is mainly because each cluster is only partially occupied and no longer fully utilized.

KB Size (nodes)	Response Speedup	Execution Speedup
1400	12.8	5.9
1600	11.9	6.1
1800	12.2	6.6
2200	12.1	7.0
2400	12.8	7.7
2800	12.7	8.2
3200	13.1	8.8
3600	13.5	9.3
4000	13.6	9.6
5000	14.2	10.6
5500	14.8	11.2
6500	14.9	11.7
7500	15.5	12.4
9000	16.0	13.2

Fig. 15. Speed-up with increasing knowledge base size.

However, if a proportionally larger knowledge base is used, then it is possible to take advantage of the parallelism, and higher speedups can be obtained. This effect is shown in Fig. 15 for a 16-cluster configuration. The speedup over a single TMS320C30 for response time and execution time increase as the knowledge base grows.

To further increase the speed performance we are developing a semantically-driven allocation scheme which assigns nodes from one concept sequence to the same processing element. This distributes the available parallelism evenly while reducing communication overhead associated with marker-propagation. In addition to improving the propagation phase, it is possible to optimize the non-propagation SNAP instructions and further reduce the total execution time. Additionally, the full 32 cluster configuration should provide a significant performance improvement for large knowledge bases.

IV. Conclusion

We have developed an integrated speech understanding algorithm based on memory-based parsing and parallel marker-passing schemes. We have provided a parallel solution to sp-

eech-specific problems such as insertion, deletion, and substitution. In our approach, meaning representations constructed as a result of memory-based parsing can be applied not only to generate an output sentence but also to provide information for applications such as high-level inferencing and speech translation. The experimental results demonstrate the benefits of the parallel computational model for the integration of speech and NLP.

References

[1] 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.

[2] S. Chung and D.I. Moldovan, "Modeling Semantic Networks on The Connection Machine," *Journal of Parallel and Distributed Computing*, Vol 17 No. 1 & 2, pp. 152-163, February 1993.

[3] R. DeMara and D.I. Moldovan, "The SNAP-1 Parallel AI Prototype," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 4 No. 8, pp. 841-854, 1993.

[4] E.P. Giachin and C. Rullent, "A Parallel Parser for Spoken Natural Language," *Proceedings of IJCAI*, pp. 1537-1542, 1989.

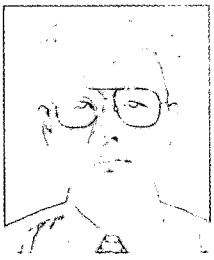
[5] X. Huang and L. Guthrie, "Parsing in Parallel," *Proceedings of COLING-86*, pp. 140-145, 1986.

[6] 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.

[7] W.S. Meisel, M.P. Fortunato, and W.D. Michalek, "A Phonetically-Based Speech Recognition System," *Speech Technology*, pp. 44-48, Apr/May 1989.

[8] L. Olorenshaw, "Air Traffic Control Training Using Continuous Speech Recognition and the ATCOACH," *Proceedings of Speech Tech '90*, pp. 376-379, 1990.

[9] D.L. Waltz and J.B. Pollack, "Massively Parallel Parsing: A Strong Interactive Model of Natural Language Interpretation," *Cognitive Science* 9, pp. 51-74, 1985.



Sang-Hwa Chung was born in Pusan, Korea. He received the B.S. degree in electrical engineering from Seoul National University in 1985, the M.S. degree in computer engineering from Iowa State University in 1988, and the Ph.D. degree in computer engineering from the University of Southern California in 1993. From 1993 to 1994, he was an Assistant Professor in Electrical and Computer Engineering Department at the University of Central Florida. He is presently a Full-time Instructor in the Department of Computer Engineering at Pusan National University. His research Interests include parallel algorithms and systems for artificial intelligence, information retrieval, speech understanding, and natural language processing.