A Parallel Computational Model for Integrated Speech and Natural Language Understanding

Sang-Hwa Chung, *Member, IEEE,* Dan I. Moldovan, *Senior Member, IEEE,* and Ronald F. DeMara, *Member, IEEE*

Abstract— **We present a parallel approach for integrating speech and natural language understanding. The method emphasizes 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, substitution, and word boundary detection have been analyzed and their parallel solutions are provided. Results on the SNAP-1 multiprocessor show an 80% sentence recognition rate for the Air Traffic Control (ATC) domain. Furthermore, speed-up of up to 15-fold is obtained from the parallel platform which provides response times of a few seconds per sentence for the ATC domain.**

Index Terms— **Speech understanding, parallel computational model, marker-passing architecture, integrated speech and natural language understanding, memory-based parsing.**

I. Introduction

A KEY issue in spoken language processing is the integration of speech and natural language understanding. Their effective integration offers higher potential recognition rates than recognition using work-level knowledge alone. Without higher-level knowledge, error rates for even the best currently available systems are fairly large. For example, CMU's Sphinx system [11] 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 70.6% for speaker-independent, continuous-speech recognition with a vocabulary of 1000 words, when recognizing individual words without using syntax or semantic information [11]. With a word recognition rate of only 70%, the overall sentence recognition accuracy will be quite low.

Clearly, we need to apply high-level knowledge to better understand continuous speech. The integration of speech and natural language understanding resolves multiple ambiguous hypotheses using syntactic, semantic and contextual knowledge sources. Since this requires sizable computations involving multiple levels of knowledge sources, speed performance can degrade considerably on realistic knowledge bases suitable for broad, complex domains. Therefore, an integrated system implemented on uniprocessor environment will face a scalability problem in the event that insufficient processing power is available as the knowledge base size is increased.

In this paper, we describe how the scalability problem can be addressed in an integrated Parallel Speech understanding System called PASS. PASS employs a memory-based parsing model and parallel marker-passing schemes to combine several levels of knowledge sources. Within this paradigm, algorithms are provided for major speech-specific problems such as insertion, deletion, substitution, and word boundary detection through tightly-coupled interaction between low-level phoneme sequences and higher-level concepts. Beyond simply recognizing speech and converting it into text, PASS employs the underlying meaning representation through parallel speech understanding. We describe an operational implementation and analyze its performance on a real parallel machine.

Manuscript received October 13, 1992; revised April 22, 1993. This work was supported by the National Science Foundation under Grant MIP-90/09109.

D. I. Moldovan is with the Department of Computer Science and Engineering, Southern Methodist University, Dallas, TX 75275–0122.

S. Chung and R. F. DeMara are with the Department of Electrical and Computer Engineering, University of Central Florida, Orlando, Fl 32816-2450.

IEEE Log Number 9213759.

II. Background

PASS uses a semantic network knowledge representation, parallel marker-passing, and memory-based parsing to effectively integrate speech and natural language understanding.

A. Semantic Networks

Semantic networks have been frequently used to represent and process structural knowledge [5]. In PASS, the semantic network representation is based on five knowledge representation elements: *concept nodes*, *relation links*, *node colors*, *node values*, and *link values*. Each domain concept is represented by a distinct node in the semantic network. Inter-concept relations are represented by directed links between nodes. The node color is used to indicate the node type. Values provide a strength indicator for relations and nodes to serve as a measure of belief during multiple hypotheses resolution.

B. Parallel Marker-Passing

A semantic network knowledge base is well suited to inferencing mechanisms based on *marker-passing* [8]. Markers are data patterns associated with each node and act as dynamic agents of inference to exchange information. Markers are used to represent properties of nodes, membership in different sets, and reflect the state of a hypothesis as they travel in parallel through the semantic network.

Whenever a marker encounters new nodes, it may change the state of knowledge associated with these nodes. Complex reasoning operations can be achieved by controlling the movement of markers through the semantic network as determined by *propagation* rules that are attached to each marker. Propa-

gation rules allow markers to individually select which paths to follow and those to avoid. To quantify properties, markers also carry a floating-point weight that is used to evaluate alternative hypotheses encountered during processing.

C. Memory-Based Parsing

Marker-passing techniques provide a reasoning mechanism for memory-based parsing approach. Memory-based parsing emphasizes a large case memory over sophisticated parsing rules or grammars. Parsing is viewed as a memory-intensive process which identifies patterns in the memory network to provide an interpretation based on embedded syntactic and semantic relations. Processing is performed using a large number of markers which propagate concurrently through the memory network. Due to the use of a large semantic network and many markers, this approach is amenable to parallel computer systems.

Memory-based parsing can be considered as an application of memory-based reasoning [17] and case-based reasoning [15] to language understanding. This view, however, differs from the traditional idea of extensive rule application to build up a meaning representation. Some models have been proposed in this direction, such as direct memory access parsing (DMAP) [16] and DMTRANS [19]. For arguments concerning the advantages of memory-based parsing, see [10] and [16].

D. Related Work

Previous research efforts have been made towards integrating linguistic information with speech recognition using various levels of knowledge sources. Paeseler [14] applied chart-parsing algorithm to speech recognition. Seneff [18] developed a probabilistic syntactic parser (TINA) for speech understanding systems. Hayes *et al.* [7] used a semantic case frame approach for parsing spoken language. Young et al. [21] proposed an integrated system (MINDS) in the context of a problem solving dialogue. MINDS used a variety of pragmatic knowledge sources to dynamically generate expectations of what a user is likely to say. Kitano [10] applied memory-based parsing in a Japanese-to-English dialog translation system.

The majority of research on parallel parsing emphasizes written language, although some recent work has addressed integrated speech understanding. Huang and Guthrie [9] proposed a parallel model for natural language parsing based on a combined syntax and semantics approach. Waltz and Pollack [20] investigated parallel approaches under paradigms related to the connectionist model.

Giachin and Rullent [6] implemented a parallel parser for spoken natural language on 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. While considerable progress has been made in speech understanding, the state of the art is still far from providing real-time speech understanding on broader domains.

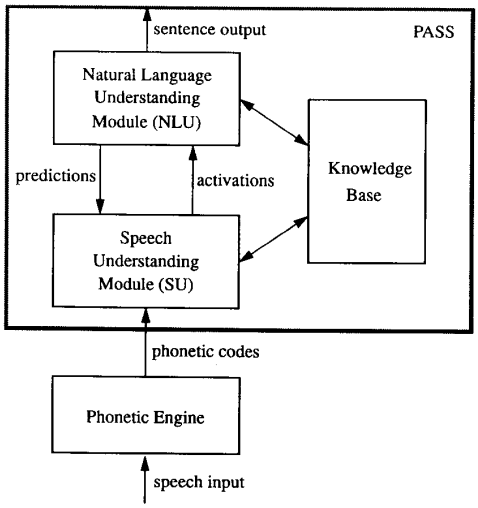

Fig. 1. The PASS environment.

III. Integrated Speech and Natural Language Understanding

PASS understands speech using techniques based on parallel processing.

A. 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 [12] manufactured by Speech Systems Incorporated. The Phonetic Engine provides a stream of input speech codes for processing. It can perform signal processing on speaker-independent continuous speech in real-time.

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.

Fig. 2 shows a part of a knowledge base from the Air Traffic Control (ATC) domain for training air traffic controllers [13]. `"Tiger six sixteen, reduce speed to one five zero"` is a typical target sentence in the ATC domain. We have adapted the ATC domain to support a vocabulary of approximately 200 words using a hierarchical semantic network of approximately 1400 nodes.

The knowledge base in Fig. 2 is organized hierarchically. Each *concept sequence* represents the underlying meaning of a possible phrase or sentence within a domain. In each concept sequence, concept element nodes are connected by *first*, *next*, and *last* links, such as the nodes in the sequence *tiger-616*

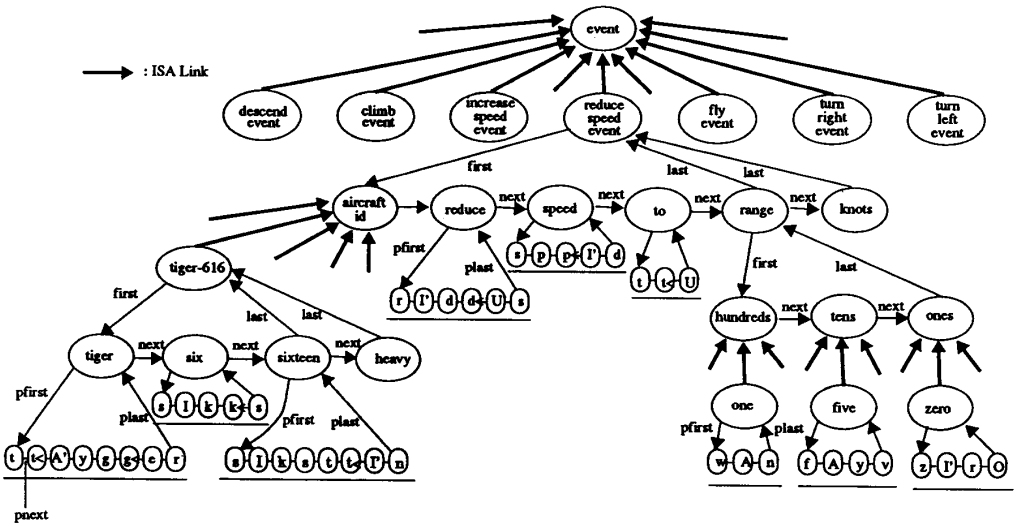

Fig. 2. Hierarchical knowledge base including phoneme sequence level—ATC domain.

and *reduce-speech-event* as shown in Fig. 2. Similarly, in each *phoneme sequence*, phonemes are connected by *p-first*, *p-next*, and *p-last* links to form words such as *tiger*. More general concept sequences are placed at higher levels, and more specific concept sequences are placed at lower levels. This type of memory network is called a *concept sequence hierarchy*. Phoneme sequences, which are attached to the corresponding concept nodes, reside at the lowest level of the concept sequence hierarchy.

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.

B. Speech-Specific Problems in Phoneme Sequence Recognition

Automated understanding of natural continuous speech presents some unique problems in addition to those already seen in written natural language.

- The influence of surrounding vowels, consonants and stress patterns can lead to the *insertion* and *deletion* of segments from the expected acoustical characteristics. Input codes may even have substituted phonemes when compared to an ideal transcription which contains only expected phonemes. For example, some elements in the sequence of input codes may be a spurious result of incorrect segmentation (insertion). Also, some of the expected phonemes may be missing as a result of incorrect segmentation, or as a result of being omitted by the speaker (deletion). In addition, a poorly articulated segment can be identified as a different phoneme type (substitution).
- The strings of input codes contain *word boundary* problems. Often words are coarticulated such that their boundaries become merged and recognizable. The phrase, `six sixteen`, is an example where the two words overlap without a clear boundary.

C. The Alignment Scoring Model

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 that 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 speech input codes generated by the phonetic engine which map to 49 phonemes. Each input code is assigned by an integer between 0 and 1643.

This 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 $S = \{c(i) : i = 0, \cdots, N\}$. To find the sentence that produced S, the memory network is searched for a sentence transcription $T = \{p(j) : k = 0, \cdots, M\}$ consisting of phonemes, each labeled $p(j)$. The correspondence of S to T that maximizes the alignment score is chosen as output.

A subsequence of codes $\{c(i) : i = i_0, \cdots, i_n\}$, $i_0 \leq i_n$, can be time-aligned with a single phoneme $p(j)$. Conversely, a subsequence of phonemes $\{p(j) : j = j_0, \cdots, j_n\}$, $j_0 \leq j_n$, 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 $c(5)$.

To compute the alignment score between S and T, score values are computed for the time-aligned subsequences of S

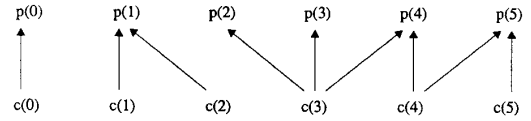

Fig. 3. A possible alignment between input codes and a target transcription.

and T. The model accounts 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, three matrices are required.

- 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 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 $p(0)$.

Insertion, deletion, and substitution can be handled in terms of the alignment scoring model [1]. 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.

D. The Functional Structure of PASS

The system organization is shown in Fig. 4. 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 partitioned and distributed to several processing elements. The scoring information including X, Y, and Z matrices is maintained in a host machine or central controller.

The algorithm is based on a combination of top-down prediction and bottom-up activation. Top-down prediction locates candidates to be evaluated next. Bottom-up activation locates possible sets of phonemes from the given input codes. As shown in Fig. 4, a circular path exists between the NLU module and the SU module: word prediction → phoneme prediction → phoneme activation → word boundary detection → word activation → multiple hypotheses resolution → word prediction. The operation starts in the NLU module by predicting the first words in possible concept sequences. This in turn impacts the prediction of the first phonemes for these words. Next, the system accepts an input code as speech input, and by consulting the X, Y, and Z matrices all 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 moves 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. Top-down prediction and bottom-up activation are performed on this circular path until all input codes are processed.

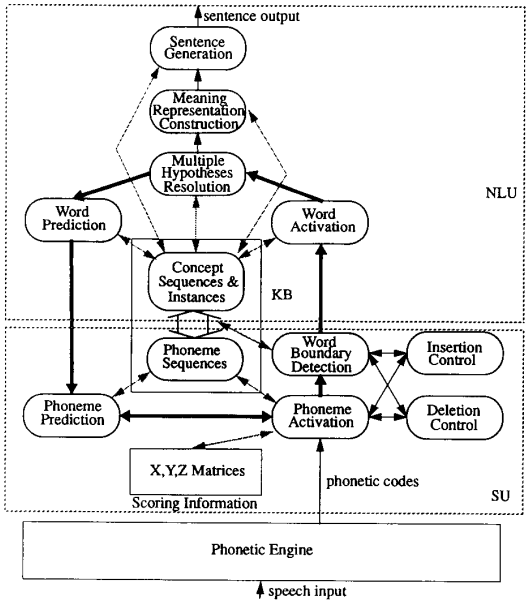

Fig. 4. Modules within PASS.

To implement the above algorithm using marker-passing, we need approximately 20 different types of markers. Both *fat-markers* and simple *bit-markers* are used. Fat-markers carry scoring information as they move through the memory network while bit-markers only convey the set membership characteristics of nodes. Some important far-markers include the following.

- *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 *p-next*) link.

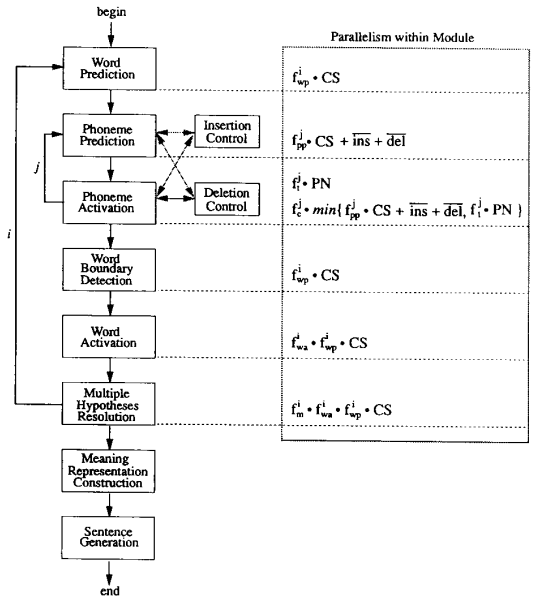

Fig. 5. Program flow and parallelism in PASS.

- *A-markers*—indicate activated 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 canceled nodes. Because of multiple hypotheses, some I-markers may be canceled or invalidated later on as their scores become inferior.

The motivation for a marker-passing approach is to exploit the concurrency in the speech understanding algorithm. Fig. 5 shows the execution flow and parallelism obtained. The amount of parallelism depends on the size of the knowledge base and the degree of ambiguity in the input code sequence. The key variables are listed in Table I. The knowledge base size is described by CS which denotes the number of concept sequences and PN which denotes the number of phoneme nodes in the memory network. The parameters $\overline{ins}$ and $\overline{del}$ correspond to extra operations introduced by insertions and deletions, respectively. Finally, the various fractional parameters in Table I correspond to each phase of the algorithm. They represent the fraction of possible concurrent operations that actually occur during each phase of Fig. 5 as described next.

1) *Word Prediction*—P-markers are propagated to words which can possibly appear next in the concept sequence hierarchy. During word prediction, $f^i_{wp} \cdot CS$ nodes are activated.
2) *Phoneme Prediction*—P-markers flow down the hierarchy to the phoneme sequence level. For the initial prediction, the first phoneme in the phoneme sequence is predicted. As processing continues, subsequent phonemes within the sequence become predicted. Thus the parallelism is determined by the number of phonemes predicted ($f^j_{pp} \cdot CS$) and the average number of operations introduced by insertions and deletions. The parallelism is dominated by the number of concept sequences rather than the number of phoneme sequences. In particular, phoneme sequences attached to corresponding concept words in each concept sequence cannot be evaluated at the same time, and processing is guided by concept sequences.
3) *Phoneme Activation*—For each input code, phoneme types in the X matrix are compared with a predetermined threshold. This determines the phonemes to be activated with A-markers. Initially, a large fraction of them are be activated in parallel ($f^j_t \cdot PN$), but many are irrelevant and rapidly eliminated. Candidates remaining are those that were predicted and activated (PA-collision). Thus, the available concurrency will approach the lesser of the number of predicted and activated phonemes ($\min\{f^i_{pp} \cdot CS + \overline{ins} + \overline{del},\ f^j_t \cdot PN\}$) multiplied by f^j_c that is the fraction of phonemes with PA-collisions.
4) *Insertion/Deletion Control*—The prediction window is adjusted to handle insertions and deletions. The window size is adjusted for phoneme sequence candidates containing phonemes with PA-collisions. The insertion/deletion control is performed along with phoneme prediction and phoneme activation for the jth iteration based on the codes received.
5) *Word Boundary Detection*—When P-markers reach the last phonemes of phoneme sequence candidates, possible word boundaries are detected. After the inner loop is completed, processing returns to the word level with the same degree of parallelism available as in word prediction.
6) *Word Activation*—A-markers are propagated up through *p-last* links to the corresponding concept sequence nodes. An I-marker is placed on the concept sequence element to indicate that the node is a possible instance and will be canceled later if not selected. Words predicted but not activated are eliminated. The fraction of predicted and activated words, denoted by f^i_{wa}, remain as parallel candidates.
7) *Multiple Hypotheses Resolution*—A-markers are propagated up through the concept sequence hierarchy to the

TABLE I
PARAMETERS AFFECTING CONCURRENCY

Parameter	Description	Observed Value
CS	total number of concept sequences	45
PN	total number of phoneme nodes	957
$\overline{ins}$	average number of operations introduced by insertions per iteration	2.3
$\overline{del}$	average number of operations introduced by deletions per iteration	1.1
$0 \le f^i_{wp} \le 1$	fraction of concept sequences active during word prediction	0.22 (avg.)
$0 \le f^i_{wa} \le 1$	fraction of concept sequences active during word activation	0.71 (avg.)
$0 \le f^j_{pp} \le 1$	fraction of phoneme sequences active during phoneme prediction	0.37 (avg.)
$0 \le f^j_t \le 1$	fraction of phonemes above threshold	0.39 (avg.)
$0 \le f^i_c \le 1$	fraction of (predicted, activated) phonemes with PA-collision	0.67 (avg.)
$0 \le f^i_m \le 1$	fraction of candidates remaining after multiple hypo. resolution	0.65 (avg.)

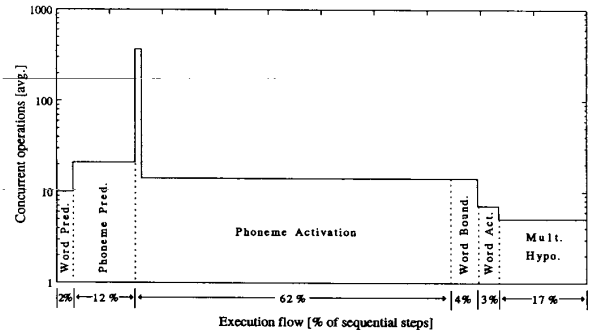

Fig. 6. Parallelism for sample target sentence.

concept sequence roots and their subsuming concepts. When multiple A-markers arrive at the same node, the concept sequence candidate with best score is selected. Steps 1–7 are once for each word activation. After the ith iteration, the fraction of multiple candidates remaining is f_m^i.

8) *Meaning Representation Construction/Sentence Generation*—For the selected concept sequences, instances are generated using the existing I-markers. The new concept sequence instances record a particular occurrence of a generic concept sequence, and together they form the semantic meaning representation. A natural language sentence is generated by analyzing the semantics of all concept sequence instances. For details of meaning representation, see [2].

Through experiment, we also measured the parallelism available by counting the actual number of activations for each type of marker. Results are shown in Fig. 6 and listed in Table I. Data was collected for the algorithm on a phase-by-phase basis for speech input codes corresponding to an entire target sentence. The results are presented for the ATC domain with $CS = 45$ and $PN \simeq 1000$. The vertical axis in Fig. 6 indicates the average number of concurrent operations available within each module over the sentence as a whole. An average value is reported since many iterations of i and j are required and the fractional parameters vary with each iteration. The horizontal axis indicates the relative number of processing steps required for each module, based on its critical path of computation. For this small knowledge base, the average parallelism ranged from 8 to 20 concurrent operations. A large spike of over 300 operations occurs at the beginning of each iteration of phoneme activation because many candidates are initially active, but then quickly fail the PA-marker collision test. Overall, about 62% of the execution flow is spent in phoneme activation which has sustained parallelism of degree 14.

Larger knowledge bases provide proportionally more concurrency. In particular, knowledge bases up to 7-fold larger size (9000 nodes) had fractional parameters close to those shown in Table I. If these parameters are relatively constant then the available parallelism will be proportional to the number of concept sequences and phoneme nodes. Thus parallelism will increase along with knowledge base size as

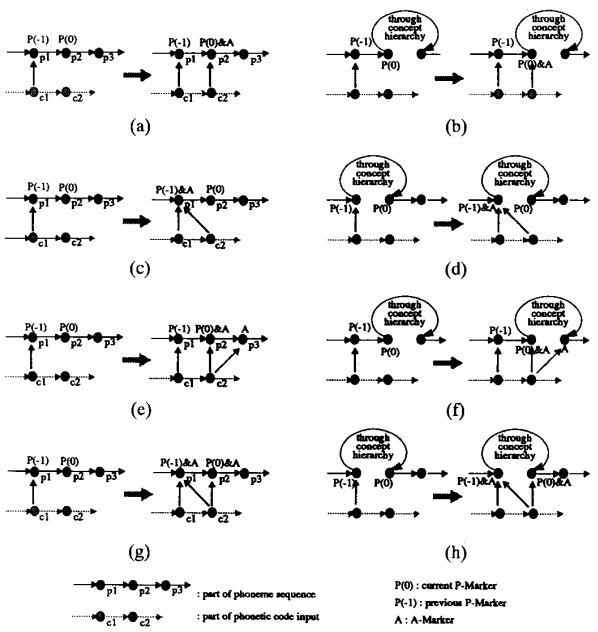

Fig. 7. Alignments including insertion and deletion problems. (a) Normal alignment. (b) Normal alignment with word boundary problem. (c) Insertion. (d) Insertion with word boundary problem. (e) Deletion. (f) Deletion with word boundary problem. (g) Insertion and deletion. (h) Insertion and deletion with word boundary problem.

described by the equations in Fig. 5. In contrast with a sequential implementation, the weaker activated candidates do not need to be pruned early in the evaluation process. This can further improve recognition accuracy by providing a chance to recover from early expectations that may later turn out to be incorrect. Through experiment, a compromise can be obtained between the parallel resources required, execution speed, and recognition accuracy.

E. Marker-passing Solutions for Insertion, Deletion and Substitution Problems

Using the X, Y, and Z matrices of the alignment scoring model, a sequence of input codes must be aligned with multiple phoneme sequence transcriptions. However, multiple candidates should be evaluated at the same time and subsequent phoneme activations cannot be foreseen while the current input code is being processed. Therefore, when we advance the prediction markers based on the current scoring information, we must consider how to recover from bad expectations in some phoneme sequence candidates, without affecting the good expectations in other candidates.

Various alignment examples are illustrated in Fig. 7. The left-hand side describes the state before a new code is processed, and the right-hand side describes the state right after phoneme candidates have been activated by the input code. Dual prediction markers, $P(-1)$ and $P(0)$, are used to keep the previously used P-marker as well as the current P-marker. Fig. 7(a) shows a normal alignment where the collision between $P(0)$ and A exists. In this case, we can normally

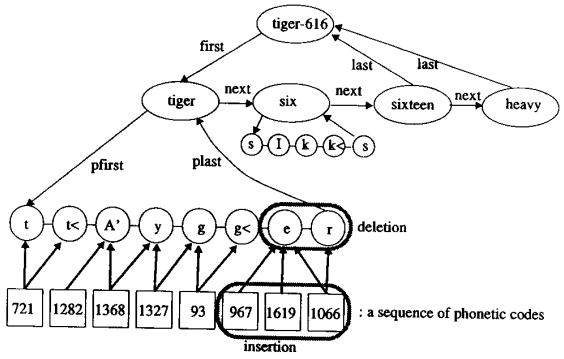

Fig. 8. Handling insertion and deletion problems.

advance $P(-1)$ and $P(0)$ to phonemes $p2$ and $p3$ to predict the next phoneme activations.

Dual predictions are required to handle the insertion problem. When there exists a strong activation (an activation beyond a threshold) to a phoneme with $P(-1)$, we regard the current code input as oversegmented. Fig. 7(c) shows the insertion problem. In this case, we cannot advance $P(0)$ to phoneme $p3$ because no collision exists between $P(0)$ and A in phoneme $p2$. However, a new $P(0)$ will be sent to phoneme $p2$ from phoneme $p1$ to reflect the newly calculated score by adding the score of the A-marker.

An example of the deletion problem is shown in Fig. 7(e). Two consecutive phonemes, $p2$ and $p3$, are activated together from code $c2$. But, phoneme $p3$ does not contain a prediction marker. In this case, the code input is regarded as undersegmented. The single code input covers phonemes $p2$ and $p3$. Therefore, $P(-1)$ and $P(0)$ can be advanced two steps through *p-next* links to predict the next phoneme activations.

In Fig. 7(g), we show a complicated alignment where both the insertion and deletion problems exist. Two consecutive phonemes, $p1$ and $p2$, are activated together from code $c2$. First, we detect the insertion by the collision of $P(-1)$ and A. When a new $P(0)$ is sent to phoneme $p2$ to update the score, we detect a deletion by the collision of $P(0)$ and A. The deletion implies that there exists no more insertion to phoneme $p1$. Thus, we can advance $P(-1)$ and $P(0)$ one step to cover the deleted phoneme $p2$, and predict the next phoneme activations.

An example of handling insertion and deletion problems is shown in Fig. 8, where only a part of the concept sequence hierarchy is depicted. We illustrate the time alignment between the subsequence of input 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`. Assume that we have processed the subsequence of codes: `721 1282 1368 1327 93`. The remaining codes can be handled as follows.

1) Code *967* is now produced from the phonetic engine. By consulting the X matrix, 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 *p-next* 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 so 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 canceled 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. New, 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*.

Substitution problems are not shown in Fig. 8. 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 later when other hypotheses get better scores.

F. 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. When no coarticulation exists between two consecutive words, it is the same as either the normal alignment of insertion problem, except that $A\&P$ Markers are moving through the concept sequence hierarchy instead of just through the *p-next* link. When a coarticulation exists between two consecutive words, it contains deletion or combined deletion/insertion problems. Some examples are shown in Fig. 7(b), (d), (f), and (h).

We handle the word boundary problem with the aid of high-level information embedded in concept sequence hierarchy. An example describing the coarticulated phrase `six sixteen` is shown in Fig. 9. 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 Fig. 9, 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

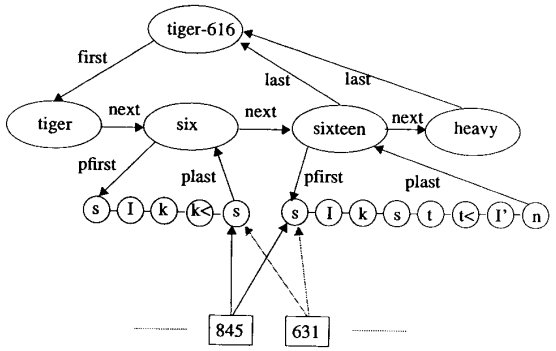

Fig. 9. Word boundary problem.

twice, reflecting newly calculated scores. As a result, $P(-1)$ and $P(0)$ are located on the first and second phonemes of concept `sixteen`. The next input code *631* activates those two phonemes on the boundary again as shown. 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.

The last phoneme of each phoneme sequence contains an L-marker indicating word boundary. During the alignment process, a word boundary can be detected by a $P\&A\&L$ collision. When a word boundary is detected, $A\&P$ Markers need to travel through the concept sequence hierarchy. In the case of `six sixteen`, markers need to move through the *p-last, next*, and *p-first* links. When a word boundary problem exists across two different concept sequences, markers need to move multiple steps up and down through the concept sequence hierarchy depending on where those concept sequences are located on the hierarchy.

G. Multiple Hypotheses Resolution

Multiple hypotheses are usually generated due to the stochastic nature of phoneme recognition errors. Thus, a list of candidates is hypothesized with different scores. Multiple hypotheses cannot be completely resolved with the information available at the phoneme sequence level. For example, consider understanding the following sentences where [ACID] denotes an arbitrary aircraft ID:

- "[ACID], `continue climbing angle`,"
- "[ACID], `continue climbing heading`,"
- "[ACID], `continue timing bases`,"
- "[ACID], `continue timing series`."

In this case, the words climbing and timing will produce similar speech code sequences from the Phonetic Engine. Once these words are accepted as candidates, then four multiple hypotheses remain. Since those two candidates are key words for the given sentences, the correct sentence understanding depends on how well the low-level ambiguities are resolved using higher-level knowledge. Specifically, as additional words are received, the range of words which are semantically valid for last word in the sentence acts to guide the resolution of the ambiguous speech data from `climbing` and `timing`. When the last word is encountered, only the score of the appropriate concept sequence will be increased.

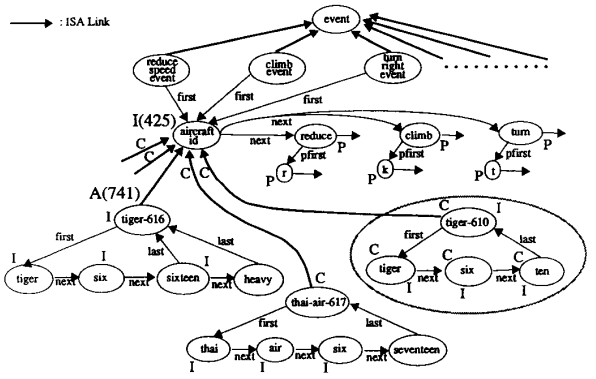

Fig. 10. Multiple hypotheses resolution.

The SU module activates multiple competing words, or 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 exist at any merging point. A merging point exists where a concept node contains multiple incoming *last* links, *isa* links, or *next* links. A merging point also exists where a concept node has multiple *p-last* links coming from multiple phoneme sequences representing various pronunciations of the concept node.

When multiple candidates arrive at a merging point, the concept node 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 canceled 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.

In Fig. 10, the concept *aircraft id* was previously activated by the hypothesis: `tiger six ten`, and the score of the I-marker for the node is the value 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 threshold can be activated from an input code regardless of its meaning. So, it is critical to set the threshold for each level of the hierarchy through experiment to prevent unnecessary activations without losing meaningful information.

When a new hypothesis: `tiger six sixteen` arrives at the node *aircraft id* with the score of the A-marker equal to a value of 741, the previous hypothesis is rejected because of its lower score value of 425. 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 canceled. 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

Code	Activations relevant to the current hypothesis (with score)		New predictions after processing current activations		
			P(-2)	P(-1)	P(0)
240	#(69)			#	t
721	t(16)	t<(22)	t	t<	A'
1282	A'(54)	y(-40)	A'	y	g
1368	A'(77)	y(34)	A'	y	g
1327	y(88)	g(18)	y	g	g<
93	g(89)	g<(66)	g	g<	e
967	e(11)	r(-34)	e	r	s
1619	e(28)	r(75)	e	r	s
1066	e(32)	r(0)	e	r	s
845	s(14)	I(12)	s	I	k
631	s(82)	I(-25)	s	I	k
857	s(86)	I(-19)	s	I	k
989	I(95)	k(-38)	I	k	k<
239	k(104)	k<(-37)	k	k<	s
576	k<(25)			k<	s
834	k<(36)	s(11)	k<	s	E'

Fig. 11. Triple prediction window.

of evaluation as shown, while later, it may turn 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 *reduce, 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.

H. Triple Prediction Window

As we discussed before, the insertion problem requires that two consecutive phonemes in a phoneme sequence be predicted at the same time. We have realized that dual prediction is insufficient when the deletion and insertion problems are intertwined. For instance, Fig. 11 shows a new scenario for Fig. 8.

When code *967* is processed, we observe that both phoneme e and phoneme r are activated together. The activation of phoneme r is very weak with the score of -34. This was not shown in Fig. 8. Actual phoneme activations depend on how we set the threshold value for the phoneme level of the hierarchy. The deletion handling routine regards this as a possible deletion problem and advances P-markers two steps. However, when code *1619* is processed, we observe that instead of phoneme s, the previous two phonemes e and r are activated with strong scores. An insertion to the second previous phoneme e exists and the deletion assumed before is wrong. As shown in Fig. 11, a $P(-2)$ keeps the P-marker for the second previous phoneme, and the system can recover from the wrong expectation using the $P(-2)$. By the prediction with a window size of 3, the combined problem of deletion and insertion can be handled elegantly.

Score of A-Marker After Activation			New Prediction Window* After Adjustment		
Phoneme L with P(-2)	Phoneme M with P(-1)	Phoneme N with P(0)	Phoneme L	Phoneme M	Phoneme N
High**	High	High	P(-1)	P(0)	P(0)
High	High	Low**	P(-1)	P(0)	
High	Low	High	P(-1)	P(0)	P(0)
High	Low	Low	P(-1)	P(0)	
Low	High	High		P(-1)	P(0)
Low	High	Low		P(-1)	P(0)
Low	Low	High			P(0)
Low	Low	Low			P(0)

* Prediction Window: P(-2) on Phoneme L, P(-1) on Phoneme M, P(0) on Phoneme N.
** Score(A): Low if Score(A) < 0, High if Score(A) >= 0

Fig. 12. Expectation adjustment based on the activation score by A-marker (subphoneme sequence: L-M-N).

I. Expectation Adjustment

Sometimes, activations do not conform to expectations. Whenever a new code activates a set of phonemes in the phoneme sequences, we need to adjust the previous expectations based on the current activation scores. The expectation adjustment table for the sub-phoneme sequence: L-M-N is shown in Fig. 12. The activation scores are scaled into the range of an 8-bit integer between -128 and 127. The score of an A-marker is high if $Score(A) \geq 0$. When the activation scores for the prediction window are {`High, High, Low`}, or {`High, Low, Low`}, we assume that the previous expectation for a deletion was not correct, and adjust the prediction window as shown. When both an insertion and a deletion occur together with high scores like {`High, High, High`}, or {`High, Low, High`}, neither one can be ignored. In this case, we partition the prediction window and keep both possibilities, as shown in Fig. 12. That is, we make two separate prediction windows for the phonemes $L\&M$ and the phoneme N in the same phoneme sequence. In most cases, the inferior one will be eliminated after one or two more input codes are processed.

IV. Execution Results

The PASS algorithm has been implemented on the SNAP-1 parallel machine and is operational. We describe below the system configuration and analyze its performance.

A. SNAP-1 Multiprocessor

SNAP-1, is a parallel array processor designed for semantic network processing with a reasoning mechanism based on marker-passing [4]. As shown in Fig. 13, the SNAP-1 architecture is based on a multiprocessing array and a dual-processor array controller. The array stores a semantic network of up to 32K nodes and 320K links. The SNAP-1 array consists of 144 Texas Instruments TMS320C30 DSP microprocessors which

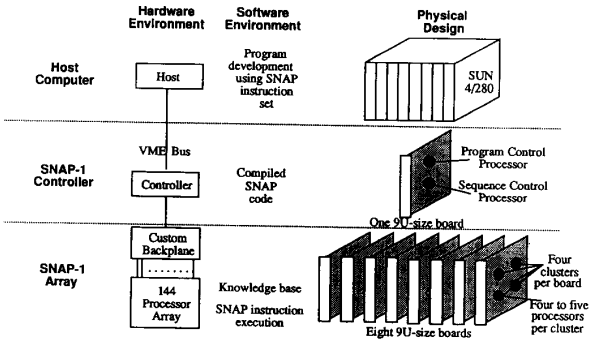

Fig. 13. SNAP-1 parallel processing system.

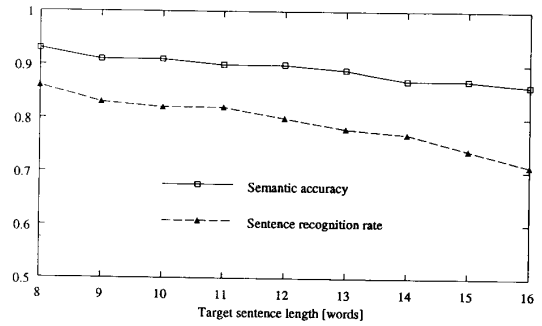

Fig. 14. Accuracy versus sentence length.

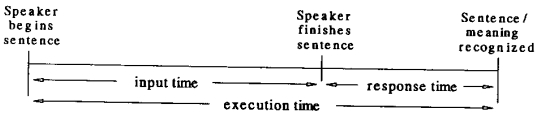

Fig. 15. Definition of execution time and response time.

act as processing elements (PE's). The array is organized as 32 tightly-coupled *clusters* of 4 to 5 PE each.[0] Each cluster manages 1024 semantic network nodes.

The controller interfaces the array with a SUN4/280 host where application programs are written and compiled in SNAP-C using libraries 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. The majority of computation is performed locally though the propagation of markers within the cluster. Several instructions and multiple propagations can be performed simultaneously to explore multiple paths in parallel.

B. System Performance

We analyzed the execution of PASS using the ATC domain with various quantities of concept and phoneme sequences. The basic ATC domain consists of 1357 semantic network nodes with 5834 links. Results obtained demonstrate the benefits of the integrated parallel model.

Recognition Accuracy: We tested 60 different continuously-uttered sentences. Each sentence contained from 8 to 16 words and was spoken by four untrained speakers. As shown in Fig. 14, a sentence recognition rate of about 80% was obtained with performance decreasing slightly as the sentences became longer and more complex. Although the overall sentence recognition rate was only 72% for sentences with 16 words, these longer sentences are encountered less frequently than shorter ones which tend to occur more often.

Furthermore, Fig. 14 indicates that the majority of sentence-level failures had nearly correct semantic meaning representations. The underlying semantics of the input sentence could be determined successfully even for long sentences. Even though one or two words may be incorrect, the speaker's intent is successfully determined according to the high-level information in the knowledge base.

Response Time and Scale-up: We 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. 15.

[0]Presently, 16 clusters are implemented in the full 5 PE configuration while the remaining 16 clusters have 4 PE's each, totaling 144 PE's.

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. 16 shows response time as a function of the total number of semantic network nodes. The dotted line is for 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 solid line in Fig. 16 is for the identical algorithm on a single TMS320C30 processor at 25 MHz. Response time also increases linearly, but the user must wait over 30 seconds for a response, even when using the basic ATC domain. The lines shown have been fitted to the measured data with slopes computed as 0.0024 and 0.04 for parallel and serial execution, respectively. Thus, while both curves increase linearly, the proportionality constant for the uniprocessor is 17-times greater, and a 15-fold speed-up is obtained from the parallel implementation for 9K nodes.

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 that 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

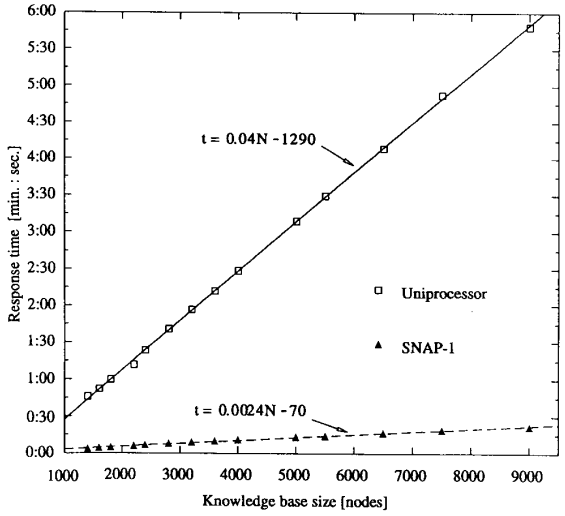

Fig. 16. Response time.

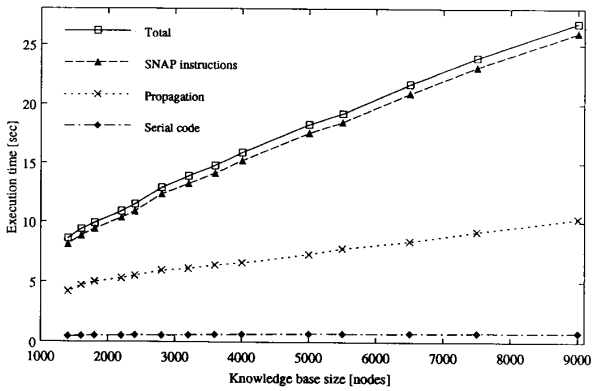

Fig. 17. Execution time.

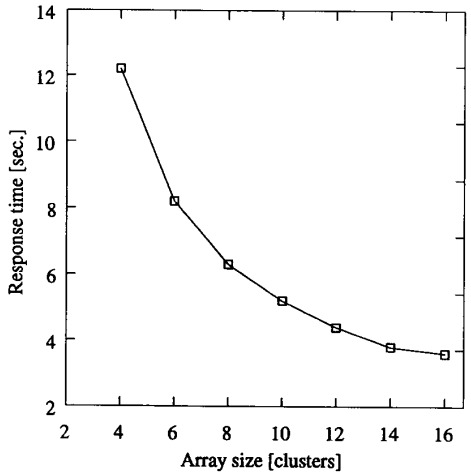

Fig. 18. Response time versus array size (knowledge base (KB) size: 1.4K nodes).

[3]. 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.

To understand the execution characteristics of the algorithm, we also studied the number and type of instructions required to process a typical target sentence. Fig. 17 shows the components of execution time on SNAP-1. The dashed line is for all 26 types of SNAP instructions, including marker-propagation. The dotted line shows that 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.

Processor Speed-Up: Fig. 18 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, execution 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 because each cluster is only partially occupied and no longer fully utilized.[1] 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. 19 for a 16-cluster configuration. The speedup over a single TMS320C30 for response time and execution time increase as the knowledge base grows.

Input Size Performance: The effect of target sentence length on response time is shown in Fig. 20 for the basic ATC domain. Although each speaker generates a varying number of speech codes, the performance is roughly proportional to the number of phonemes in the target sentence. Each sentence has an average of 11 words or about 50 phonemes long and takes about 3 to 5 seconds until the meaning representation and output sentence is generated. To further increase the speed performance for larger knowledge bases, the full 32 cluster configuration will be used when it becomes available.

[1] With the 1.4K basic ATC domain, the average parallelism ranges from 8 to 20, while 40 PE's are available in a 16 cluster configuration. Other 32 PE's are dedicated to program control, communication, etc. For details of cluster design, see [4].

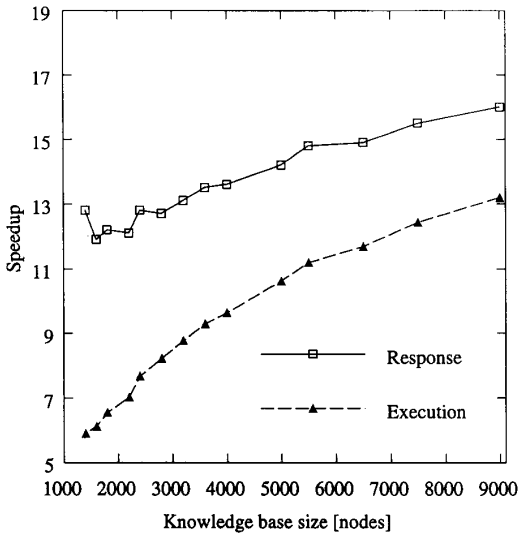

Fig. 19. Speed-up with increasing knowledge base size.

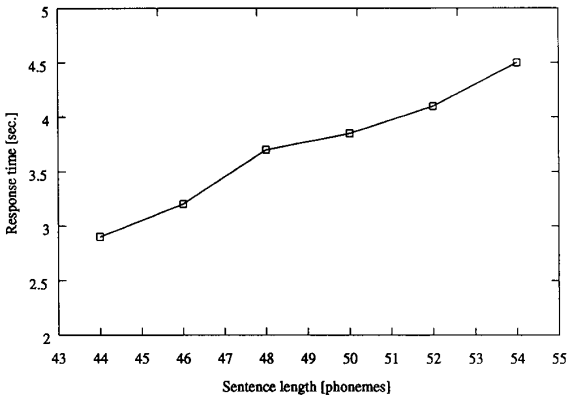

Fig. 20. Response time versus target sentence length on 16 clusters.

V. Conclusion

We have developed an integrated approach for speech understanding from low-level speech input up through meaning representation and sentence generation. Using marker-passing techniques, we have presented parallel solutions to insertion, deletion, substitution, and word boundary problems. For limited domains, near real-time performance has been obtained with a speed-up of up to 15-fold over a sequential implementation. The meaning representations which are generated 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 computation model for the integration of speech and natural language understanding. We are now installing the Radiology domain[2] provided by the Speech Systems Incorporated consisting of a vocabulary of approximately 2-K words and a network of approximately 10-K semantic nodes.

[2]The radiology domain supports interactive document preparation for radiologist when dictating reports based on examination of an X-ray.

References

[1] M. T. Anikst and D. J. Trawick, "Training continuous speech linguistic decoding parameters as a single-layer perceptron. *Proc. Int. Joint Conf. on Neural Networks,* vol. 2, pp. 237–240, 1990.

[2] M. Chung and D. I. Moldovan, "Memory-based parsing on SNAP: Integrated syntactic and semantic analysis," Tech. Report PKPL 91-10, Dept. of Elect. Eng. Syst., Univ. of Southern California, Los Angeles, 1991.

[3] ——, "Modeling semantic networks on the connection machine," *J. Parallel and Distributed Computing,* vol. 17, pp. 152–163, Feb. 1993.

[4] R. F. DeMara and D. I. Moldovan, "The SNAP-1 parallel AI prototype," presented at the *Proc. 18th Ann. Int. Symp. on Comput. Architecture,* Toronto, ON, Canada, 1991.

[5] S. E. Fahlman, *NETL: A System for Representing and Using Real-World Knowledge.* Cambridge, MA: MIT Press, 1979.

[6] E. P. Giachin and C. Rullent, "A parallel parser for spoken natural language," *Proc. IJCAL,* 1989, pp. 1537–1542.

[7] P. J. Hayes, A. G. Hauptmann, J. G. Carbonell, and M. Tomita, "Parsing spoken language: A semantic caseframe approach," *Proc. COLING-86,* 1986, pp. 587–592.

[8] J. A. Hendler, *Integrating Marker-Passing and Problem Solving: A Spreading Activation Approach to Improved Choice in Planning.* Hillsdale, NJ: Lawrence Erlbaum Associates, 1988.

[9] X. Huang and L. Guthrie, "Parsing in parallel," *Proc. COLING-86,* 1986, pp. 140–145.

[10] H. Kitano, "ΦDM-Dialog: A speech-to-speech dialogue translation system," *Machine Translation,* vol. 5, pp. 301–338, 1990.

[11] K. F. Lee, "Large-vocabulary speaker-independent continuous speech recognition: The sphinx system," Techn. Rep. CMU-CS-88-148, Dept. of Comput. Sci., Carnegie-Mellon Univ., Pittsburgh, PA, 1988.

[12] W. S. Meisel, M. P. Fortunato, and W. D. Michalek, "A phonetically-based speech recognition system," *Speech Technol.,* Apr./May 1989.

[13] L. Olorenshaw, "Air traffic control training using continuous speech recognition and the ATCOACH," presented at *Proc. Speech Tech '90,* 1990.

[14] A. Paeseler, "Modification of Earley's algorithm for speech understanding," *Recent Advances in Speech Understanding and Dialog Systems.* Berlin: Springer-Verlag, 1987.

[15] C. K. Riesbeck and R. Schank, *Inside Case-Based Reasoning.* Hillsdale, NJ: Lawrence Erlbaum Associates, 1989.

[16] C. K. Riesbeck and C. E. Martin, "Direct memory access parsing," Tech. Rep. YALEU/DCS/RR #354, Dept. of Comput. Sci., Yale Univ., 1985.

[17] C. Stanfill and D. Waltz, "Toward memory-based reasoning," *Commun. ACM,* vol. 29, no. 12, Dec. 1986.

[18] S. Seneff, "TIMA: A probabilistic syntactic parser for speech understanding systems," *Proc. DARPA Speech and Natural Language Workshop,* 1989, pp. 168–178.

[19] H. Tomabechi, "Direct memory access translation," *Proc. IJCAI,* 1987, pp. 722–725.

[20] D. L. Waltz and J. B. Pollack, "Massively parallel parsing: A strong interactive model of natural language interpretation," *Cognitive Sci.,* vol. 9, pp. 51–74, 1985.

[21] S. R. Young, A. G. Hauptmann, W. H. Ward, E. T. Smith, and P. Werner, "High level knowledge sources in usable speech recognition systems," *Commun. ACM,* vol. 32, no. 2, pp. 183–193, Feb. 1989.

Sang-Hwa Chung (S'92) 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.

His research interests include parallel algorithms and systems for artificial intelligence, speech understanding, and natural language processing. He is presently an Assistant Professor in Electrical and Computer Engineering Department, University of Central Florida, Orlando, FL.

Dan I. Moldovan (S'79–M'78) was born in Sibiu, Romania. He received the M.S. and Ph.D. degrees in electrical engineering and computer science from Columbia University, New York, NY, in 1974 and 1978, respectively.

He was a member of the Technical Staff at Bell Laboratories from 1976 to 1979, after which, he took on a position as an Assistant Professor of Electrical Engineering at Colorado State University from 1979 to 1981. He was a member of the faculty of Computer Engineering at the University of Southern California from 1981 to 1993. Presently, he is a Professor of Computer Science and Engineerin and Director of the Parallel Computers Research Laboratory at Southern Methodist University, Dallas, TX. He took a one year sabbatical leave from 1987 through 1988 to work at the National Science Foundation in Washington, DC, as Program Director for Experimental Systems in the Division of Microelectronics and Information Processing Systems. His primary research interests are in the fields of parallel processing and artificial intelligence, in which he's published over 100 papers.

Professor Moldavan is the author of the textbook, *Parallel Processing: From Applications to Systems* (Morgan and Kaufmann, 1993).

Ronald F. DeMara (S'87–M'93) received the B.S. degree in electrical engineering from Lehigh University, Bethlehem, PA, in 1987 and the M.S. degree in electrical engineering from the University of Maryland, College Park, in 1989 and the Ph.D. degree in computer engineering from the University of Southern California, Los Angeles, in 1992.

From 1987 through 1989, he was an Associate Engineer at IBM Corporation, in Manassas, Virginia where he was involved in the design of embedded and complex systems. He is currently an Assistant Professor in the Electrical and Computer Engineering Department at University of Central Florida, Orlando, Florida. His research interests are in the areas of parallel processing, artificial intelligence, and computer architecture.