SPEECH RECOGNITION IN PARALLEL

Salvatore J. Stolfo, Zvi Galil, Kathleen McKeown and Russell Mills
Columbia University
Department of Computer Science
New York, N.Y. 10027

INTRODUCTION

Concomitantly with recent advances in speech coding, recognition and production, parallel computer systems are now commonplace delivering raw computing power measured in hundreds of MIPS and Megaflops. It seems inevitable that within the next decade or so, gigaflop parallel processors will be achievable at modest cost. Indeed, gigaflops per cubic foot is now becoming a standard of measure for parallel computers.

Now that affordable massively parallel computing is approaching reality, it is conceivable that raw computing power can bring gigaflops to bear on the speech recognition problem. Experiments can now be undertaken in a more timely manner and systems can be organized, perhaps with improved recognition accuracies. It is our present thesis that speech recognizers can be improved, not only in response time but in reduced error rates as well by parallel computing. How?

We are investigating parallel algorithms that combine a number of concurrent and independent acoustic processors and speech recognizers that may, we conjecture, synergistically deliver better overall recognition performance than any component in isolation. Simply stated, our approach is to utilize a number of concurrent and competing recognizers in the aggregate, utilizing much more information from the speech signal than has been attempted before in one individual recognition system. Thus, rather than committing a single recognition system to model and recognize phones and words from a particular set of information, processed from the raw acoustic signal, we conjecture that utilizing much more of the information available from the signal may effect better overall recognition performance. We aim to compose multiple independently-executing recognizers into one recognition system through trained, weighted voting schemes.

Our sights are aimed at the lower level of the recognition process: better phone recognition leading to improved word recognition. Clearly, the general problem of speech recognition requires "higher order" information to completely recognize sequences of words and sentences and their semantics. Such knowledge-based approaches serve to bias the recognition process in favor of "sensible" word utterances and to disambiguate word utterances by reducing the size of the search space of possible words at each point of an utterance through language, pragmatic and semantic constraints. We suspect, however, that such higher order information can be rendered more effective if the lower level recognizer can achieve near perfect recognition of the phones in the utterance initially. That is, if the ordering of likelihoods of candidate phones and, ultimately, words, are wrong, or worse if the wrong set of candidates is posited, no amount of higher-level knowledge will allow the recognition of an utterance with high reliability, unless a language model over-constrains the problem. (Clearly, as a consequence, utterances governed by low perplexity grammars are easier to recognize than ones with higher perplexity grammars.)

The first aim of our work will be to produce more accurate phone, and thus word, recognition. Isolated word recognition stands the most to gain initially from our approach. However, if successful, the approach will allow higher levels of knowledge to more effectively do the job of word sequence recognition and understanding in connected and continuous recognition tasks.

We plan to test the interaction of higher level knowledge with lower level recognition by incorporating a syntactic parser also implemented using parallel algorithms and hardware. In later stages of our work we shall incorporate higher level constraints from syntax, and possibly semantics, through the addition of a separate independently-executing recognizer that will vote on the likelihood of the next word based on syntactic and semantic expectations. We will use a functional unification grammar [Kay 79, McKeown & Paris 87, Elhadad 89] for this task because it is suited for representing complex interaction between multiple often conflicting constraints, it will easily lend itself to parallelization, and it will allow for later extensions to speech synthesis quite easily. This higher level recognizer will be incorporated into the speech system later in our proposed research plan. In the initial phase of our work, we will focus on the adaptation of the grammar for interpretation (we have been using it solely for generation) and on the parallelization of the unification process.

This approach can certainly be pursued with a serial computing system simply by expanding the set of features extracted from the speech signal to include much more information from the acoustic signal in the first place and executing each recognition task sequentially in turn. However, the computation required for a serial approach expands in an amount proportional to the number of features extracted, and the number of recognizers. The problems of realizing realtime performance, as well as reducing the overhead computational cost of model training, thus become exacerbated.

Processing each recognizer in parallel with the others, however, results in a system no slower than the slowest individual recognizer. Furthermore, if each recognizer itself can be executed as a parallel activity, realtime performance might be achievable with the current generation of hardware. This approach, therefore, calls for computing structures with many processing elements, typically called massively parallel computers.

Our aim, however, is to design massively parallel computing structures that are economical; minimum parallel computing requirements may be met by simple parallel computing structures that scale economically.

Another approach we are pursuing is to explore recent advances in dynamic programming algorithms for sequence matching tasks that have been shown to reduce the time of conventional serial algorithms by as much as an order of magnitude (see [Eppstein 89a] for example). Hence, it may be possible that realtime performance of speech recognition may be approached with faster serial pattern matching algorithms. Furthermore, if we find the means of efficiently parallelizing these newer serial algorithms, realtime recognition might be directly achieved.

By way of summary, we attack the problems of realtime speech recognition with improved recognition accuracy by:

- utilizing much more information from the raw acoustic signal,
- composing multiple recognition systems into one aggregate recognizer through trained, weighted voting schemes,
- using higher level syntactic, and possibly semantic, constraints for speech recognition through the incorporation of an additional recognizer using natural language approaches,
- exploiting recent algorithmic advances in dynamic programming,

- parallelizing the dynamic programming algorithms and multiple recognizer paradigm,
- and demonstrating these features on a speech recognition system running on economical and scalable parallel hardware.

In our ongoing work, we have investigated and implemented parallel computing systems to run pattern recognition algorithms at high speeds. The research devoted to the multiple recognizer paradigm is in its initial stages of inquiry. Presently, we are completing experimental parallel hardware, with operational software systems to be ported from an earlier version of the machine, to conduct a set of experiments. This work will be outlined later.

MULTIPLE SPEECH RECOGNIZERS COMBINED INTO ONE

One serious problem with most speech recognition systems is that the output from the acoustic preprocessor is a small set of numbers that carry too little information about the original speech signal. To address this problem, researchers have begun examining how additional extracted information improves the recognition process

RELATED APPROACHES

For example, K.-F. Lee [Lee 88] has recently constructed at Carnegie-Mellon University the most accurate large-vocabulary continuous-speech recognizer so far. Starting with a baseline system based on hidden Markov models, he made a succession of improvements to its acoustic-modeling component that significantly increased its recognition accuracy. Simply adding another set of acoustic parameters provided the greatest improvement in recognition accuracy; using triphone models to model the coarticulatory effects of a sound's context on that sound provided the next greatest improvement. There is clearly room for further improvement in acoustic modeling.

A number of inaccuracies in the assumptions underlying most Markov models of speech further distort and blur the information that remains. P. Brown [Brown 87] has recently conducted experiments that seem to indicate that continuous-parameter distributions can model speech more accurately than vector-quantized feature vectors. Continuous-parameter distributions require vast amounts of training data, however, and there is little known about the shape of the actual distribution of speech sounds.

Even though the acoustic analysis of speech has received a great deal of attention (see [Makhoul 85]), there is little agreement about the best set of acoustic parameters to use. Many sets have been proposed and tested, and include

- Energy at different frequencies in the speech signal;
- Time-domain characterizations of the waveform, such as zero crossings, or zero crossings in various frequency bands;
- Productive models, such as models of the vocal tract as a set of resonant tubes;
- Perceptual models, based on physical models of the cochlea and psychoacoustic experiments.

In fact, few hard data are available about the relative strengths and weaknesses of the different parameter sets. Little published data exists on systems that have attempted recognition utilizing combinations of different acoustic parameter sets.

Several systems have tried to combine a single set of parameters with information about their time derivatives; some

of these are described below.

One system using more than one parameter set is described by [Tribolet 82]. This isolated-word system based on dynamic time warping used long-term LPC coefficients in slowly-changing sections of speech and short-term LPC coefficients in unvoiced fast-changing segments; the categorization of sections of speech as slowly-changing or fast-changing depended on a comparison of the short-term and long-term LPC coefficients. Adding short-term coefficients to a baseline system based on long-term coefficients contributed very little to recognition accuracy. The authors suggest that short-term LPC coefficients were an inappropriate choice of parameters for the rapidly-changing unvoiced sections of speech, and that the large steady-state regions in vowels swamped the contributions to the distance computation of the few transient frames.

More recently, K.-F. Lee [Lee 88] has used multiple sets of parameters (measurements of energy in 12 frequency bands, differenced measurements of energy, total power, and differenced power) quantized with three disjoint codebooks to increase recognition accuracy. In his system, each frame of parameters consisted of an entry from each codebook, which the system treated as uncorrelated. Thus the probability distribution on each transition consisted of the product of the distributions for each codebook. Using three separate codebooks led to much lower quantization error and higher recognition accuracy than concatenating the sets of parameters and using a single codebook. The three sets of acoustic parameters he combined were all computed in the same fixed-width time windows. He also attempted to combine parameters computed in variable-width windows with fixed-width parameters; this approach was *ad hoc* and led to little increase in recognition accuracy.

P. Brown [Brown 87] has concatenated acoustic parameters from adjacent time frames in a recognition system using continuous distributions and full covariance matrices. He applied discriminant analysis to reduce the number of parameters to train. His approach led to an improvement in recognition accuracy on the E-set (the easily confused "e"-sounding letters of the alphabet).

COMBINING INDEPENDENT RECOGNIZERS

We are attacking the problems of low recognition accuracy and overwhelming computational demands by combining multiple independently-executing recognizers into one through trained, weighted voting schemes. Several sets of recognizers might be constructed from different sets of acoustic parameters, or from different procedures for constructing codebooks. In the ideal case, the recognizers so combined would have independent (uncorrelated) error patterns, so that a simple vote could reduce the overall error rate dramatically. In such a case, if p is the error rate of the worst of n recognizers, the "combined" recognizer would be no worse than:

$$\binom{2n-1}{n} p^n (1-p)^{n-1}$$

Since the errors made by the recognizers will undoubtedly be correlated, albeit imperfectly, a simple vote will not suffice. In essence, we seek statistically sound methods to build into our combined recognizer the independence of the component recognizers needed to approach the ideal case.

Evidence of the soundness of this approach has been provided by [Stephanou 88]. In that paper, the authors studied the combination of two inherently incomplete knowledge sources that reach a consensus opinion using Dempster-

Shafer theory in a problem-solving setting. They rigorously prove that the entropy of the combined consensus system is less than the entropy of each individual system. Thus, theoretically, errors made by the combined system should appear less frequently than either constituent.

Initially, we are studying the correlations among a number of different acoustic parameter sets and different codebooks, both in a purely statistical fashion and in terms of the errors made by recognizers built on the parameter sets. A recent paper by Gillick and Cox [Gillick 89] provides a principled means of determining the statistical significance of the differences among alternative recognition systems. Then, using a reasonable set of maximally independent parameter sets and codebooks, we will construct combined recognizers that use the remaining correlations (inverted) as weights for voting.

Voting among recognition systems based on hidden Markov models can be incorporated as part of the traceback procedure. In a single recognizer, the Viterbi dynamic-programming search algorithm constructs a matrix that for each state and input frame tells the most likely state at the previous input frame. The traceback procedure begins at the final state of the network and the final input frame and, proceeding backwards in time, constructs the most likely path through the network. In combining multiple recognizers, we propose to maintain for each recognizer several back pointers, together with a rating of each, at each state and input frame, and to trace back all the models together, pooling the information from all the recognizers.

One major question about the procedure will be the granularity of the voting scheme. During the traceback, do the various recognizers vote on individual words, on phonemes within words, or on analysis frames within phonemes? The frequency of voting may affect recognition accuracy as well as the decomposability of the recognition procedure.

This is a key technical problem to be studied in this research. At present we have some idea of general approaches that may have utility. For example, symbolic AI machine learning techniques may be exploited to learn associational rules that infer likely recognized phones or words from contextual information provided by multiple recognizers trained for different units of speech. The output of a single recognizer of some interval of speech is typically a list of rank ordered candidate phones, that is, a list of symbols representing the modelled phones believed to be present in the interval of speech. Two recognizers operating on the same interval of speech would produce two lists of symbols (presuming, of course, both recognizers can be time aligned with accuracy). These two lists of symbols may be viewed as two "representations" of the same speech source, or rather two sets of "hypotheses" of a particular speech utterance. The task is then to define a method of combining these two bodies of evidence to reach a consensus on what was uttered.

Suppose we have two recognizers, one based on diphone models, the other on triphones. Rules may be learned during training that associate the appearance with some diphone, for example, appearing in one list with some triphone model appearing in the other list. One idea is to use the list of triphone candidates generated during recognition along with the associational rules to reorder the list of candidate diphones from the other recognizer. During recognition, each applicable associational rule can be applied by scanning a triphone candidate list generated by one recognizer and adding to a "frequency count" to each associated diphone appearing in the candidate list of the other recognizer. The end result would be a reordered candidate list from the diphone-based recognizer, ranked according to maximum frequency count.

It is difficult to propose a cogent argument for the utility of this approach without first performing a number of preliminary experiments. For example, it might be more useful to reorder the candidate list by using the normalized frequency counts as weights applied to the distances of the initial candidate diphones. Issues of time alignment, integration of multiple models and possible exponential growth in numbers of rules that may be generated, must be studied first with specific examples. The point is, however, that there is a distinct possibility that we may reduce the "voting problem" to a symbolic form suitable for applying various AI techniques. Alternatively, neural network techniques may have utility in solving this problem.

INCORPORATING HIGHER LEVEL CONSTRAINTS

Following initial experimentation with isolated word recognition, we will turn our attention to problems involved in continuous speech. It is that point that we plan to incorporate higher level constraints from syntax and semantics. These constraints can be extremely helpful in discriminating between several words when low level recognition techniques produce more than one highly ranked candidate. For example, sentential syntactic context can be used to discriminate two similar sounding words such as "form" and "farm" when appearing in a sentence such as "The clouds form a pattern that continues for miles." Our approach will be to construct a separate natural language recognizer that will vote on candidate words based primarily on syntactic expectations. While many have recognized the value of including natural language approaches in speech recognition, few systems include more than a very simple grammar (see for example [Lee 88].)

One of the problems in adapting natural language approaches for speech is that people rarely speak in complete, grammatical sentences. Instead, speech errors and colloquialisms abound. Previous approaches to dealing with ungrammatical input have focused either on systematically relaxing grammatical constraints [Weischedel 83], on developing grammars specifically for speech through analysis of transcribed speech [Hindle 89], or on the use of semantically based parsers [Lebowitz 83, Lytinen 84]. Of these, we favor relaxing constraints, primarily because large speech grammars are not currently available for general use and because we have found that syntax does provide very useful clues to interpretation.

Relaxing grammatical constraints to find an acceptable parse is a combinatorially explosive computational task, however. We have selected a *functional unification formalism*, FUF, for the task for this reason. One of its primary advantages is its ability to deal with the complex interaction of multiple constraints. Furthermore, its functional aspects will allow us to encode more than purely syntactic constraints and these can be called into service in parallel with syntax. Finally, we plan to implement a parallel version of the unification algorithm and this should also address computational issues in relaxing constraints.

We have already developed a large grammar in FUF that we are currently using for generation along with a well documented unifier that has been substantially tested, in part through class use [Elhadad 89]. To use FUF for speech recognition, our tasks for the early phase of our proposed research will include the conversion of our FUF unifier for interpretation of language. Note that this will result in a reversible syntactic processor so that we will be well situated to use components of our speech recognizer for synthesis as well. The second task will be the parallelization of the unification algorithm. Because the search through the grammar is a decomposable problem (each alternative in the set of grammar rules can be passed off to a different processor), this is clearly possible.

The final step will be the incorporation of the natural language recognizer with the remaining lower level recognizers. This will involve the development of a suitable voting scheme that can adequately weigh advice from both components.

PARALLELISM

Our study aims to balance generality with economy and performance. We expect to fall in the middle by uncovering minimal parallel processing requirements to meet the needs of the model evaluation portion of the speech recognition task; these requirements may be met by an architecture that is not fully general (implying low cost in hardware) but also not highly specialized (implying that it is not limited too severely in scope of use). We believe an architecture composed of a distributed set of processing elements (PE's), each containing local memory and high speed DSP processors, with a limited interconnection and communication capability may suit our needs. From our studies so far, we have found that a general high flux interconnect is simply not needed to speed up dynamic programming. Communication does not appear to dominate the computation involved in the various algorithms employed in speech recognition.

In our earlier work, we invented and implemented on the DADO2 parallel computer one approach to parallelizing the match task of test patterns to references by parallelizing the dynamic programming algorithm. The approach is strictly a data parallel activity. This work has been previously reported in [Stolfo 87] where we carefully compare our approach to a similar yet different approach proposed by Bentley and Kung. In [Bentley and Kung 79], they propose a systolic parallel machine, based on a "dual tree" interconnection, to rapidly execute a stream of simple dictionary-type queries. (Their machine was never realized.)

For the present paper, we briefly outline a class of searching problems that generalizes those investigated by Bentley and Kung and describe the method by which we parallelize them on the DADO parallel computer.

A *static searching problem* is defined as follows:

- Preprocess a set F of N objects into an internal data structure D.
- Answer queries about the set F by analyzing the structure D.

Following Bentley [Bentley 78], we note that in many cases, such problems are solvable by serial solutions with linear storage and logarithmic search time complexity.

The *membership problem* provides an illustration of this kind of problem. Preprocess N elements of a totally ordered set F such that queries of the form "is x in F" can be answered quickly. The common solution for serial computers is to store F in a sorted binary tree structure D and perform binary search. (Of course, hashing is common for problems where hashing functions can be defined.) Thus, the membership problem can be computed on sequential computers with logarithmic time complexity.

A *decomposable searching problem* is a searching problem in which a query asking the relationship of a new object x to a set of objects F can be written as:

$$\text{Query}(x, F) = \mathop{B}_{f \text{ in } F} q(x, f)$$

where B is the repeated application of a commutative, associative binary operator b that has an identity and q is a "primitive query" applied between the new object x and each element f of F. Hence, membership is a decomposable searching problem when cast as

$$\text{Member}(x, F) = \underset{f \text{ in } F}{\text{OR}} \text{ equal}(x, f).$$

The *Nearest Neighbor problem*, which determines for an arbitrary point x in the plane its nearest neighbor in a set F of N points, can be cast as

$$\text{NN}(x, F) = \underset{f \text{ in } F}{\text{MIN}} \text{ distance}(x, f).$$

Based on the work of Dobkin and Lipton, Bentley points out that Nearest Neighbor has log time serial complexity in searching a static data structure. The Nearest Neighbor problem is closest to problems in pattern recognition to be detailed shortly.

Decomposable searching problems are well suited to direct parallel execution. The key idea about these kinds of problems is *decomposability*. To answer a query about F, we can combine the answers of the query applied to arbitrary subsets of F. This characteristic also guarantees quick execution in a parallel environment. The idea is simply to partition the set F into a number of subsets equal to N, the number of available PE's. (For pedagogical reasons, in what follows we assume a single set element f is stored at a PE.) Apply the query q in parallel at each PE between the unknown x that is communicated to all the PE's and the locally stored set element f. Finally, combine the answers in parallel by log N repetitions of b. This last step proceeds by applying N/2 b-computations simultaneously between "adjacent" pairs of PE's. The N/2 resultant values are processed again in the same fashion. Hence N/4 b-computations are applied in parallel, producing N/8 results. After log N steps the final single result is computed.

ALMOST DECOMPOSABLE SEARCHING PROBLEMS

The approach we invented (and implemented on the DADO2 machine [Stolfo 87]) is quite different to Bentley and Kung's approach to solving decomposable searching problems as well as variations that we call *almost* decomposable searching problems. In a nutshell, queries are rapidly broadcast to all the PE's in a parallel processor. Primitive queries are executed in parallel by all PE's, and in several important cases, the combined result of applying operator b is obtained very quickly with parallel hardware support. (In the case of DADO2, this step takes one instruction cycle for up to 8,000 PE's.) We have called this mode of operation Broadcast/Match/Resolve/Report and it will be described with examples shortly.

First, we note several variations of these kinds of problems to clarify the benefits of the approach that we have taken:

1. Consider searching problems where a static data structure, D, cannot be defined. In vector quantization of some random source, or template matching in isolated word recognition, finding the best match (closest centroid of the clustered space) requires the calculation of the distance of the new unknown sample to all members of the reference set. Binary searching, for example, of the set of centroids is not possible in general.
2. Consider problems where a single query in a series of queries cannot be computed without knowing the result of the previous query. In dynamic programming approaches to statistical pattern matching tasks, a single match of an unknown against the set of references cannot be computed without knowing

the best match(es) of the previous unknown(s). Hence, for a series of unknown x_i, i=1,...,M,

$$\text{Query}(x_i, F) = \mathop{B}_{f \text{ in } F} q(x_i, \text{Query}(x_{i-1}, F), f).$$

In this case, a pipe flushing phenomenon appears forcing systolic approaches to suffer computational losses.

3. Consider problems where the "combining" operator, b, is not commutative, nor associative, but otherwise the searching problem remains quite the same. Thus, a parallel algorithm of some sort would be applied to the results of the primitive queries, q, applied to the reference set.

4. Lastly, consider searching problems where we wish to compute a number of different queries about the same unknown x over a set, or possibly different sets. Hence

$$\text{Query}_i(x, F) = \mathop{B}_{f \text{ in } F} q_i(x, f) \text{ for } i=1,...,M.$$

Our approach to combining multiple speech recognizers provides an illustration of this type of problem.

PARALLEL MACHINE REQUIREMENTS

How do we achieve high execution speed of almost decomposable searching problems on parallel hardware? We seek the most economical way of solving almost decomposable searching problems in parallel. This entails ascertaining the properties of the parallel computation in question and the parallel hardware required to perform it.

MIMD vs SIMD

Case 1 above is clearly handled by any parallel machine simply by executing matching functions, or primitive queries, in parallel. The data structure common to serial implementations is replaced by the parallel machine programmed as an "associative memory processor". However, each execution of the primitive query, q, may require different instruction streams to operate at each PE. (In our following discussion, q may be DTW matching or HMM evaluation, or both operating concurrently in different PE's.) Clearly, an SIMD parallel computer will not be effective in cases where MIMD parallelism is called for. This feature demands that each PE have significant program memory available to it rather than depending on a single broadcast stream of instructions executed in lock-step fashion. Distributed memory machines, as well as shared memory (omega network-based or bus-based) machines certainly provide this capability.

Communication Model

For case 2, the communication model of a parallel machine should support the rapid broadcast of data to all PE's in the machine. That is, communicating a single quantity of data should take O(log N) *electronic gate delays* in a parallel machine *not* O(log N) *instruction cycles* via instruction pipeline communication. DMA speeds are clearly desirable. Conversely, the "reporting" of a single datum from one PE to another should also be performed in a small amount of time. This key architectural principle allows for the rapid communication of data in and out of the machine as well as from one distinguished, but arbitrarily chosen, PE to all others. Hence, communicating a single data item from one PE to all others is achieved in time proportional to the size of the data, not O(log N) instruction time. Internally generated queries, as in case 2, can be reported and broadcast in a constant number of instruction

cycles. Pipe flushing problems as in a systolic machine need not concern us. (Below we detail the timing of broadcast rates in our experimental hardware system, currently under development, specifically in the case of broadcasting speech sample data.)

Built-in Hardware Features

Another key capability of a parallel machine to execute almost decomposable searching problems efficiently is to provide direct hardware support for quickly computing a range of commutative and associative binary operators B. In our membership problem defined above, the binary operator OR is repeatedly applied to all of the results of the primitive query "equal (x, f)". In a sequential environment, this operation may require linear time to compute. In a parallel environment it can be computed in log time. On a parallel machine with some hardware support, it may be computed in constant instruction cycle time. The I/O circuit of DADO2, for example, provides a high speed function that we call *min-resolve*. The min-resolve circuitry calculates in *one instruction cycle* the minimum value of a set of values distributed one to a PE. Not only is the minimum value reported in a single instruction cycle, but the PE with the minimum value is set to what is called the "winner state," providing an indication to the entire ensemble of loser PEs, as well as identifying the single winner PE in the computation. The membership problem can thus be solved by applying min-resolve to zeros and ones (distributed throughout the machine after complementing the result of the equality operator) to compute OR. Nearest Neighbor can be computed by applying min-resolve to the distances (one eight bit word at a time).

Min-resolve has proven to be a very useful primitive in our studies of parallel algorithms. In certain cases it is not enough. When the binary operator, b, is not commutative, nor associative, more general parallel processing is of course needed to combine the results of applying primitive queries to distributed data. In an example below, calculating the mode of a set of distributed data provides a convenient illustration of case 3.

Partitioning

Problems characterized by our fourth example can be efficiently supported by a parallel machine that can be logically partitioned into disjoint parallel computers. In this case each partition may execute a distinct task. This is a critical requirement for our proposed multiple recognizer paradigm described in detail below.

SPMD mode and Data Parallelism

Let us now review how a parallel machine can quickly execute almost decomposable searching problems. Nearest-neighbor will provide the vehicle for illustration.

1. Preprocess N objects of set F by distributing each element in turn to one PE of the machine. Repeat the following:
2. Broadcast the unknown object x to all PEs in time proportional to the size of the object.
3. Apply the query "distance (x, f)" in parallel at each PE. In parallel, each PE sets the min-resolve value to its locally calculated distance.
4. Min-resolve on the distributed set of distance scores in parallel very quickly.

Figure 1 depicts this approach in terms of a tree-structured parallel architecture.

The overall computation time for processing a query is therefore O(|x|)+ O(q) + O(1), the sum of steps 2, 3, and 4. In the sequential case, the computation would be O(|x|) + O(log|F|)O(q), the time to read the unknown, plus the time to apply the primitive query to the elements of the reference set appearing along a path through the log|F| deep data structure D, and combine the results. In cases where the data structure D cannot be defined, the serial complexity rises to O(|x|) + O(|F|)O(q), as in the case of pattern recognition tasks. Note, therefore, that the parallel running time is constant in the size of the reference set F. Scaling a recognizer to incorporate a larger reference set F implies that the parallel architecture should scale linearly to maintain constant time performance; that is, doubling the size of the reference set implies doubling the number of PE's at twice the cost in hardware.

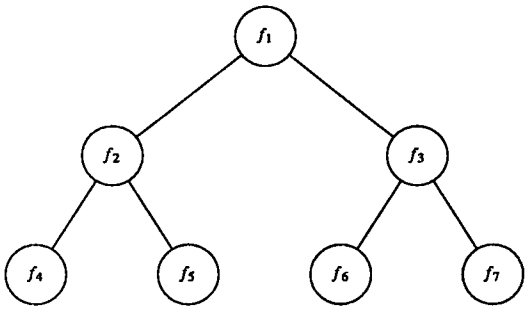
This mode of parallel operation clearly captures the familiar data parallel operations popularized by the Connection Machine and others. A pure SIMD-based machine, however, is not very well suited to executing the more general class of almost decomposable searching problems especially if the execution of the primitive query q requires alternative code sequences to be executed in each PE. In our earlier work we identified this more general parallel mode of operation as Single Program Multiple Data stream, or SPMD mode. SPMD mode is required for speech recognition tasks.

PE Architecture

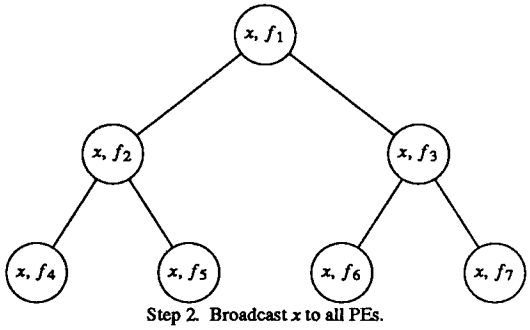
The running time of the primitive query distance, O(q), is clearly dependent on the particular distance metric computation chosen (and certainly on the size of the unknown). Generally, q can be the computation of the "best" matching word model, selected from a set of distributed (HMM) word models. The complexity is then dependent on the form of the models (the number of states in the longest left-right HMM, for example). In our membership example above, the primitive query "equal" takes constant time. Similarly, in Nearest-Neighbor, distance is assumed also to take constant time. In comparing analysis frames to acoustic models in speech recognition, however, the primitive query calculates a distance between two acoustic feature vectors in the case of codebook computations, or the best word model accounting for a sequence of observation vectors, requiring many floating point operations. Thus, our parallel machine's performance depends greatly on how fast a PE can calculate this function, and hence we explicitly show O(q) in our running time complexity.

Simple one-bit PE's would struggle to execute these functions in realtime. This is an important consideration. A particular processor implementation may take 10 times as long to execute q as another. A parallel processor consisting of processors of the former type would need ten times as many PE's to compete with another parallel processor consisting of PE's of the latter. The choice of PE processor is driven by the frame broadcast rate. A frame may be broadcast every centisecond. Hence, a PE must calculate the distance function, q, communicate results and update paths in the search space within this time. This argues for very fast floating point hardware at each PE. In our prototype hardware described below, we describe the use of fast DSP chips as the main computational engine of a PE and the resultant computing cycles available for speech processing.

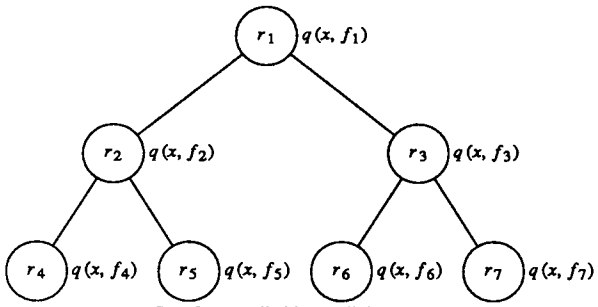
Figure 1. A general parallel solution of almost decomposable queries.



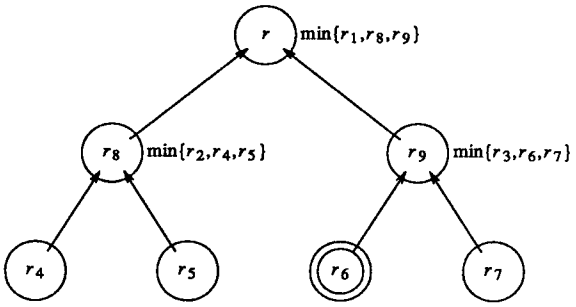
Step 1. Set elements distributed.



Step 2. Broadcast x to all PEs.



Step 3. q applied in parallel.



Step 4. Minimum value r calculated at root.
The PE with value r is set (represented by a double circle).

SPEECH RECOGNITION AS AN ALMOST DECOMPOSABLE SEARCH PROBLEM

Now we can clearly state the advantages to speech recognition; dynamic time warp serves as our example. During the course of executing the dynamic time warp algorithm a set of quantized vector codes, (F in our example above) distributed to each PE is matched in parallel against the current frame of speech (x_i in our examples above). Prior to the broadcast of the next query (frame x_{i+1}) some number of the best results (shortest distance vector codes) are reported and broadcast to all PEs which then update their current set of best paths, as maintained by the dynamic time warp algorithm. The next frame, x_{i+1}, is then broadcast and the cycle repeats itself until a final posited word is recognized, or the end of the utterance is reached.

Figure 2 graphically depicts the familiar general speech recognition architecture in terms of an almost decomposable search problem.

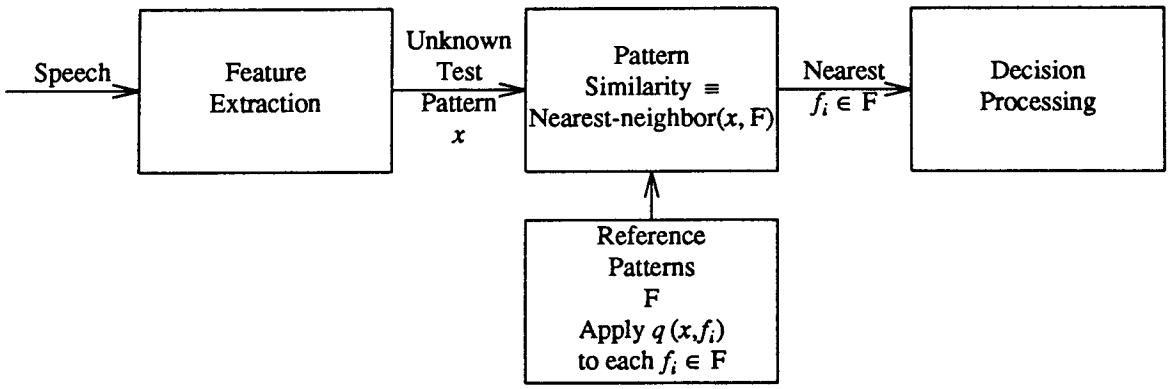


Figure 2. Speech recognition as an almost decomposable search problem.

Note that the sequence of broadcast speech frames is continually punctuated by synchronous global communication in this approach. The time available between broadcast frames is available for parallel processing and inter-PE communication. The unidirectional systolic pipelining approach is not appropriate in this context, as noted above, since this task requires bidirectional global communication that would otherwise flush the systolic pipe. Since a single "winning" PE must send its best matching phone to all other PE's, asynchronous models of inter-PE communication (message passing, for example) are not appropriate as well: if a processor completes its task early, and then communicates its result, it must sit idly waiting for the next frame of speech. There are no other speech processing tasks for it to perform. Thus, it is best that all PE's complete their tasks roughly at the same time. Indeed, we therefore require simpler communication for these tasks since we do not need message protocol processing.

Voting

To execute a number of concurrent (and independent) tasks requires partitioning whole tasks both in software and

hardware. Partitioning a parallel machine is generally straightforward, unless the computational model of the machine imposes a single locus of control for all PE's (say, for example, a single host processor broadcasting instructions to all subservient PE's). Each partition exercising its own control can thus execute its own recognizer according to the scheme outlined above. The set of reference acoustic models in each partition can be quantized vector codes of various phones (diphones, triphones, etc.), or word-based templates, or hidden Markov models of words or word sequences. The particular algorithm executed, whether it is dynamic time warp or Viterbi search, although similar in structure since both rely on dynamic programming, would operate completely independently within a single partition. In the simplest case, each partition may be a single PE executing a complete recognizer (serially, of course, but in parallel with the others). All partitions, however, would ultimately synchronize to vote on the utterance for final consensus recognition.

The voting of a number of independent recognizers is a straightforward hierarchically parallel computation. Indeed, we may cast our voting based recognizer composed of a number of dynamic time warp recognizers as an almost decomposable query as follows:

$$\text{Recognize-phone(Speech-data,Combined-recognizer)} = \underset{i}{\text{Mode}}\ \text{Query}_i(\text{Analysis-frame,Component-recognizer}_i),\ i=1,\ldots,M$$

where

$$\text{Query}_i(x_i,F_i) = \underset{f \text{ in } F_i}{\text{MIN}}\ \text{distance}_i(x_i,f)$$

M is the number of component recognizers, x_i is a particular analysis frame extracted by the i^{th} recognizer from the speech data, F_i is the i^{th} set of distributed acoustic models and distance_i is the particular distance function used for the i^{th} component recognizer.

Figure 3 depicts the organization of the multiple recognition paradigm, while figure 4 illustrates this organization as an Almost Decomposable Search Problem. Figure 4a depicts the case where each recognizer is executed wholly within one PE, while figure 4b depicts the case where a single recognizer may utilize a number of adjacent PE's.

Here we have chosen to use "mode" as our means of voting for pedagogical reasons. Choosing the most frequently posited recognized phone is only one method. This computation, unlike min-resolve, requires counting members of a distributed bag (or multiset) in parallel. High-speed communication within the machine, as well as min-resolve to choose the highest frequency datum, provides a fast way to calculate mode. For example, enumerating and broadcasting of each recognized phone followed by a frequency counting step results in a set of distributed frequencies of occurrences of individual phones. Min-resolve can then select the most frequently occurring phone very quickly. This computation takes O(M) time in this scheme, where M is the number of component recognizers. We may choose to use a sorting network-based parallel machine to reduce this computation time. Notice, however, that the calculation of mode from a relatively small number of component recognizers is supported by the fast global communication and min-resolve operations of our idealized parallel machine and is dominated by the time to calculate distances of vector codes. It nat be overkill to require a general interconnection topology for a potentially small part of the overall computation.

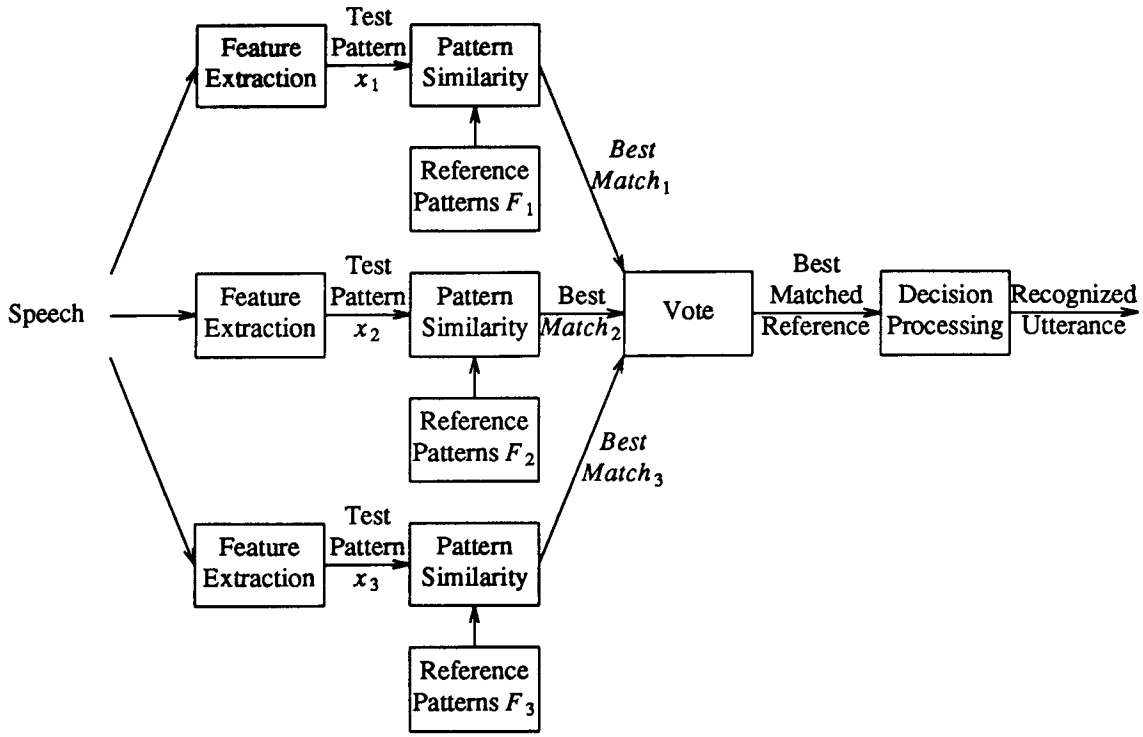


Figure 3. Organization of Multiple Speech Recognizers executed in parallel.

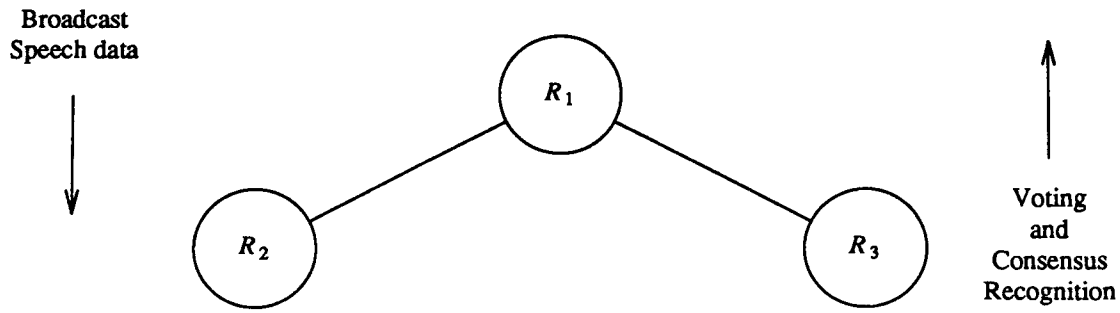
Clearly, we may choose a number of other voting schemes including majority voting, or 2 of N voting, for example. The precise voting scheme, as noted earlier, is one of the issues we are studying experimentally.

Load Balancing

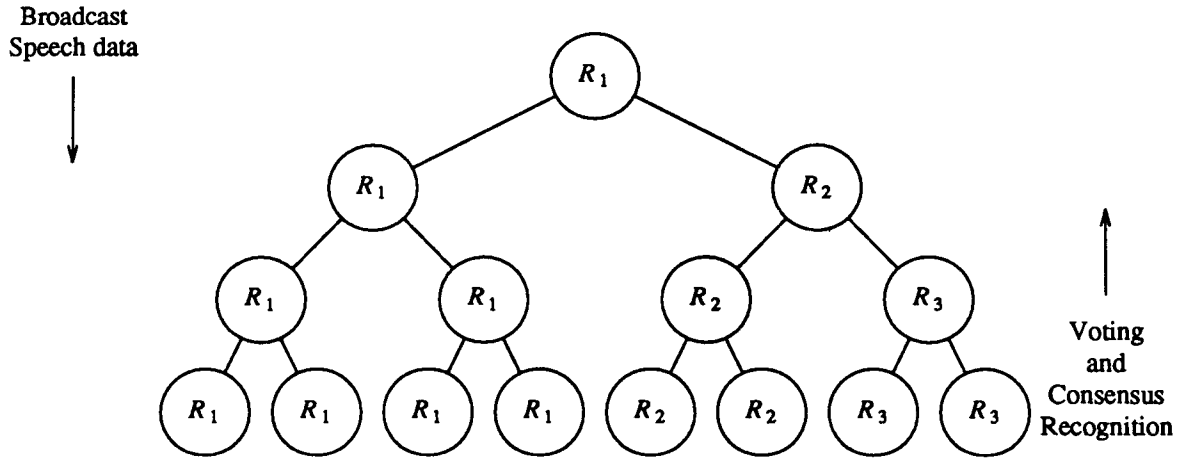
Another critical problem to study is load balancing of the parallel activities and allocating PE's to the component recognizers. Although a partitionable parallel machine may successfully implement our multiple recognition paradigm, care must be taken to ensure that the total utilization of all PEs in the system is as high as possible. Each recognizer clearly will require different amounts of computing time and is dependent on the particular distance calculation and number of models stored at each PE. Note, in our earlier discussion we presumed a signle reference (template or model) is stored at a PE. Varying the number of models at a PE varies the amount of computing time required at a PE. The single synchronization point of voting undoubtedly must be carefully orchestrated so that no single recognizer, or partition of PE's, sits waiting idly by until all others catch up for the final voting and subsequent broadcast of the next speech frame. Thus, it is important to study and automate load balancing techniques to match the computation time of each recognizer as closely as possible.

This requires automated systems to allocate perhaps different numbers of PE's in each partition. PE's in different partitions undoubtedly must store different numbers of acoustic models to balance the total computation time of each partition. Figure 4b depicts the case where each recognizer requires different numbers of PE's. No data can be

provided at this time until we actually build a number of recognizers and perform detailed timing analyses to begin solving this problem.



a) A complete recognizer in one PE.



b) A complete recognizer using adjacent PEs. Note R_1 utilizes 8 PE's, R_2 utilizes 4, while R_3 utilizes 3 PE's.

Figure 4. Organizations of a Multiple Speech Recognizer System on a tree-structured parallel machine.

IMPROVING DYNAMIC PROGRAMMING

There is another possible approach to the problem of different recognizers taking differing amounts of time, which can be applied orthogonally to the above-mentioned technique of non-uniformly partitioning the PEs. That is, developing new algorithmic techniques, to speed up the slower recognition algorithms. This also has obvious benefits to sequential as well as parallel computation of speech recognition tasks.

In particular, the time warp dynamic programming problem greatly resembles some problems of sequence alignment with convex gap cost functions [Sankoff 83], for which Zvi Galil and his students have found speed ups of as much

as an order of magnitude [Galil 89, Eppstein 89a, Eppstein 89b, Eppstein].

More specifically, the sequence alignment problems, as with the time warp problems, can be represented as filling out entries in a dynamic programming matrix of quadratic size. However, for the harder sequence alignment problems, each entry of the matrix depends on all entries in the column above it or in the row to the left of it; thus a straightforward computation of each entry would lead to a cubic time algorithm. But by taking advantage of the convexity or concavity inherent in typical gap cost functions, and by using simple data structures such as stacks and queues, the computation of all entries in a given row or column can be performed in linear or close to linear time; thus the time for the entire algorithm is quadratic or close to quadratic.

We can achieve even further speed-ups if we can determine that many entries of the dynamic programming matrix can not contribute to an optimal solution to the problem. With the help of some further algorithmic techniques, including divide and conquer as well as the recently developed monotone matrix searching technique [Aggarwal 87], the time can be reduced to almost linear in the number of remaining sparse matrix entries [Eppstein 89a, Eppstein 89b]. The details of the computation become more complicated, but this is made up for by the reduction in the size of the problem.

It seems likely that similar methods will provide practical speed ups to the time warp algorithm, bringing its complexity closer to that of the other recognition algorithms. The sparse sequence alignment technique mentioned above is especially intriguing, because the complexity introduced in dealing with sparsity resembles in certain respects that of the time warp problem; further, the fact that the sparse alignment problem can be solved efficiently gives rise to hope that the same techniques can be used to solve time warping. It is also pertinent to ask whether the time warp problem can have similar sparseness properties, and to take advantage of these in its solution.

EXPERIMENTAL HARDWARE

We are performing our studies on a small scale tree-structured machine, a newer version of the DADO machine that we call DADO4. The general parallel processing requirements of fast broadcast, report and min-resolve, as well as partitionable MIMD processing and distributed (program) memory can be provided by a bus-based architecture, be it a tree or a linear bus. Indeed, the tree structure simply provides a high-speed global communication bus and a convenient structure for partitioning the machine. No general interconnection topology is necessary to execute almost decomposable search problems and thus speech recognition tasks.

We believe tree-structures may provide the necessary communication bandwidths to deliver data to the PE's and report the final results of matching references to test patterns. For example, the prototype DADO4 is designed with 15 DSP32C chips delivering in the aggregate approximately 300 megaflops on a two board system. Each DSP32C-based DADO4 PE is clocked at 40 MHz delivering 20 Megaflops per processor. Each PE has a full megabyte of storage (and space is available on the prototype boards to increase this memory configuration if needed).

The DADO4 is designed with a new I/O feature, an advance over its earlier predecessors. The broadcast of 32-bit word data is handled through high speed PAL circuits allowing data to reach all PE's essentially instantaneously. (The use of PAL's allows us to directly experiment with other hardware I/O features that we may find useful later in our research. For example, a built-in hardware feature to allow high-speed voting of a number of recognizers may

have utility.)

Let us assume we sample speech at a 20KHz sampling rate with 16 bits per sample. That generates data at 40 Kilobytes per second. The DADO4 I/O circuits can broadcast blocks of data at 20 megabytes per second. Thus, we may broadcast a 200 sample block, or 100 words, in .002 centisecond. The percentage of time devoted to I/O for speech sampling at a high sampling rate is only .2% in this machine configuration. Each centisecond, nearly 3 megaflops is available for processing each frame of speech. Thus, nearly all of the 300 megaflops computing power of the ensemble of DSP chips is available for direct computation. This substantial number of operations is applied to acoustic preprocessing, matching and report operations.

There are other reasons why we believe a tree structured topology suffices in our experimental work. Trees are not overly specialized since such architectures have been the focus of study over the years leading to a wide range of algorithms for many problems (see [Kosaraju 89] for a recent example). Trees are especially well suited to executing almost decomposable searching problems, as noted earlier, which we believe provides a general model for many pattern recognition tasks. Trees are also partitionable: a single tree can be dynamically configured into a collection or forest of subtrees. Hence, a number of concurrent recognizers can be executed effectively on trees.

Trees are well known to be efficiently scalable [Stolfo 84]. Linear embeddings in a plane of trees argues for high density scaling and low volume devices. Constant pin out in interconnecting tree based modules argues for linear cost in scaling to larger numbers of PE's, and thus larger vocabulary speech recognizers and larger numbers of recognizers can be supported optimally. Thus, if our approach to executing multiple recognizers on tree structures succeeds, the resulting system has a significant chance of delivering realtime and low-cost dedicated speech recognition devices.

Indeed, since nearly all parallel architecture networks have an underlying spanning tree, our work should be easily generalized and immediately applicable to any MIMD parallel computer that is partitionable. This we believe is an important advantage. Since the scalability of trees is well known, it is not unreasonable to expect that if our techniques are successful, then massive parallelism for dedicated speech recognition systems can be delivered by low cost tree-structured parallel computers. Inexpensive front-end speech recognition might therefore be possible. However, other more general-purpose parallel computing systems can also deliver speech recognition using our techniques with little difficulty by a straightforward mapping of our algorithms.

CONCLUSION

By way of summary, the major goal of our research is to demonstrate that recognition accuracy and speed can be improved by composing independently-executing recognizers, and that each recognizer can itself be executed as a parallel process. Along the way, we hope to provide answers to the following questions:

- How can we use more of the acoustic information present in the speech signal? Existing speech recognizers do not use all the available information; human listeners perform much better than computers even on tasks, like the E-set, that depend only on acoustic information. We hope to demonstrate that it is possible to use more acoustic information without requiring excessive amounts of training, and to use this information in a parallelizable (hence quickly computable) way.
- How are different acoustic measurements correlated? It is unlikely that they all provide the same information to a recognizer. If they do, then it doesn't matter which set of features a recognizer uses,

but there is also something fundamentally wrong in the acoustic processing used by current recognizers, since the speech signal conveys enough information to allow people to understand speech much better so far than computers. If they do not, then we should be able to improve recognition accuracy by using more acoustic information.

- How can we merge the acoustic information derived from different units of speech with different time alignments? As K.-F. Lee has pointed out, it is difficult to combine variable-width parameters and fixed-width parameters in a single Markov model. We hope to show that it is possible to merge such information coming from different Markov models.
- How can we merge acoustic information with higher level constraints? Our approach will include the incorporation of a syntactic natural language component to discriminate among different word candidates.
- Can we speed up the computation involved in speech recognition by running parts of it in parallel in a cost effective manner? Certainly various speech recognition algorithms have been parallelized on a variety of parallel machine architectures. Can we provide the same on a tree architecture utilizing the multiple speech recognizer paradigm?
- How might we balance the load among a number of independent speech recognizers to deliver maximum performance and utilization of the parallel processor? Can the load balancing be completely automated at compile time, or are dynamic runtime facilities required?
- Can we speed up, either serially or in parallel, the DTW and Viterbi search, for example, by applying new advances in algorithms for dynamic programming?

REFERENCES

[Aggarwal 87] Aggarwal, A., Klawe, M. M., Moran, S., Shor, P., Wilber, R.
Geometric Applications of a Matrix-Searching Algorithm.
Algorithmica 2:209-233, 1987.

[Bentley 78] Bentley J. L.
Decomposable Searching Problems.
Information Processing Letters 8:244-250, June, 1978.

[Bentley and Kung 79] Bentley, J. and Kung, H.T.
A Tree Machine for Searching Problems.
Technical Report, Carnegie-Mellon University, Computer Science Department, September, 1979.

[Brown 87] Brown, P. F.
The Acoustic-Modeling Problem in Automatic Speech Recognition.
PhD thesis, Carnegie Mellon University, May, 1987.

[Elhadad 89] Elhadad, M.
Extended Functional Unification ProGrammars.
Technical Report, Columbia University, New York, NY, 1989.

[Eppstein] Eppstein, D.
Sequence Comparison with Mixed Convex and Concave Costs.
J. Algorithms , .

[Eppstein 89a] Eppstein, D., Galil, Z., Giancarlo, R., Italiano, G. F.
Sparse Dynamic Programming I: Linear Cost Functions.
J. Algorithms , 1989.

[Eppstein 89b] Eppstein, D., Galil, Z., Giancarlo, R., Italiano, G. F.
Sparse Dynamic Programming II: Convex and Concave Cost Functions.
J. Algorithms , 1989.

[Galil 89] Galil, Z., Giancarlo, R.
Speeding up Dynamic Programming with Applications to Molecular Biology.
Theor. Comput. Sci., 1989.

[Gillick 89] Gillick, L. and Cox, S.J.
Some Statistical Issue in the Comparison of Speech Recognition Algorithms.
In *Proceedings ICASSP-89*. IEEE, 1989.

[Hindle 89] Hindle, D.
Deterministic Parsing of Syntactic Non-fluencies.
In *Proceedings of the 21st Annual Meeting of the Association for Computational Linguistics*, pages . Cambridge, Mass., June, 1989.

[Kay 79] Kay, M.
Functional Grammar.
In *Proceedings of the 5th meeting of the Berkeley Linguistics Society*. Berkeley Linguistics Society, 1979.

[Kosaraju 89] Kosaraju, S.R.
Pipelining Computations in a Tree of Processor.
Proceedings of FOCS-89, October, 1989.
To Appear.

[Lebowitz 83] Lebowitz, M.
Memory-Based Parsing.
Artificial Intelligence 21(4):363 - 404, 1983.

[Lee 88] Lee, K.-F.
Large-Vocabulary Speaker-Independent Continuous Speech Recognition: the SPHINX System.
PhD thesis, Carnegie Mellon University, April, 1988.

[Lytinen 84] Lytinen S.L.
The Organization of Knowledge in a Multi-lingual Integrated Parser.
PhD thesis, Yale University, 1984.

[Makhoul 85] Makhoul, J., Roucos, S., and Gish, H.
Vector Quantization in Speech Coding.
Proceedings of IEEE 73(11):1551-1588, November, 1985.

[McKeown & Paris 87] McKeown, K.R. and Paris, C.L.
Functional Unification Grammar Revisited.
In *Proceedings of the ACL conference*, pages 97-103. ACL, July, 1987.

[Sankoff 83] Sankoff, D., Kruskal, J. B. (editor).
Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison.
Addison-Wesley, 1983.

[Stephanou 88] Stephanou, H. E., Lu, S.-Y.
Measuring Consensus Effectiveness by a Generalized Entropy Criterion.
IEEE Transactions on Pattern Analysis and Machine Intelligence 10(4):544-554, July, 1988.

[Stolfo 84] Stolfo S. J., and D. P. Miranker.
DADO: A Parallel Processor for Expert Systems.
In *Proceedings of the 1984 International Parallel Processing Conference*. IEEE, Michigan, 1984.

[Stolfo 87] Stolfo S. J.
Initial Performance of the DADO2 Prototype.
IEEE Computer Special Issue on AI Machines 20:75-83, January, 1987.

[Tribolet 82] Tribolet, J. M., Rabiner, L. R., Wilpon, J. G.
An Improved Model for Isolated Word Recognition.
Bell System Technical Journal 61(9):2289-2312, November, 1982.

[Weischedel 83] Weischedel, R.M. and Sondheimer, N.K.
Meta-Rules as a Basis for Processing Ill-Formed Input.
American Journal of Computational Linguistics 9(3-4), 1983.