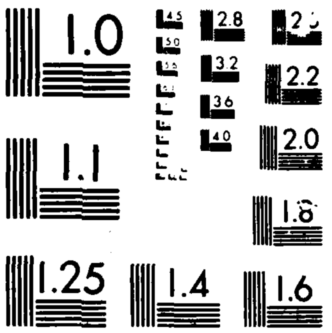
AD-A176 258 DEVELOPMENT OF REAL-TIME SPEECH RECOGNITION(U) BBN LABS 1/1
INC CAMBRIDGE MA M A KRASNER ET AL. JAN 87 BBN-6444
N00039-85-C-0313

UNCLASSIFIED F/G 9/2 NL

END
2 87
DTIC


1.0
4.5
5.0
5.6
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6

AD-A176 258

(12)

BBN Laboratories Incorporated

A Subsidiary of Bolt Beranek and Newman Inc.

Report No. 6444

DEVELOPMENT OF REAL-TIME SPEECH RECOGNITION

Item 0001: Final Report

January 1987

Prepared for:
Defense Advanced Research Projects Agency
and
Space and Naval Warfare Systems Command

87 1 28 [illegible]

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER / 2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) DEVELOPMENT OF REAL-TIME SPEECH RECOGNITION ITEM 0001: FINAL REPORT	5. TYPE OF REPORT & PERIOD COVERED Final Technical Report 3 June 1985-2 Dec. 1986
	6. PERFORMING ORG. REPORT NUMBER 6444
7. AUTHOR(s) Michael A. Krasner, Richard Schwartz, Owen Kimball, Lynn Cosell	8. CONTRACT OR GRANT NUMBER(s) N00039-85-C-0313
9. PERFORMING ORGANIZATION NAME AND ADDRESS BBN Laboratories Incorporated 10 Moulton St. Cambridge, MA 02238	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Department of the Navy Space and Naval Warfare Systems Command Washington, D.C. 20363-5100	12. REPORT DATE January 1987
	13. NUMBER OF PAGES 11
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)	15. SECURITY CLASS. (of this report) UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Distribution of the document is unlimited. It may be released to the Clearinghouse, Dept. of Commerce, for sale to the general public.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

speech recognition, continuous speech, real-time speech processing, parallel processing

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report describes our work during the first phase (Item 0001) of this contract. In this phase, we have developed parallel processing techniques appropriate for real-time speech recognition and demonstrated the feasibility of achieving real-time recognition on the BBN Butterfly™ Parallel Processor.

The speech recognition algorithm that is the focus of this effort is being developed for the BBN Byblos speech recognition system. The

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

multiprocessor used in the research, the BBN Butterfly Parallel Processor, is a shared memory, MIMD machine. The algorithm was implemented using the Uniform System software methodology, a system that simplifies parallel programming without sacrificing efficiency. The algorithm is described, highlighting those portions critical to an efficient parallel implementation. Some of the problems encountered in trying to improve the efficiency are presented as well as the solutions to those problems. The algorithm is shown to achieve 79% processor utilization on a 97-node Butterfly Parallel Processor. This is equivalent to speedup by a factor of 77 over a single processor benchmark.

Report No. 6444

DEVELOPMENT OF REAL-TIME SPEECH RECOGNITION

ITEM 0001: FINAL REPORT

January 1987

Prepared by:

BBN Laboratories Incorporated
10 Moulton Street
Cambridge, Massachusetts 02238

Prepared for:

Defense Advanced Research Projects Agency
and
Space and Naval Warfare Systems Command
Computational Systems and Technology Division (SPAWAR613)
Under Contract No. N00039-85-C-0313

Table of Contents

1. Executive Summary

The purpose of this effort is to develop parallel processing techniques for real-time speech recognition. This is part of an overall program to demonstrate the utility and effectiveness of high performance speech recognition for natural human-machine interaction.

This report describes our work during the first phase (Item 0001) of this contract. In this phase, we have developed parallel processing techniques appropriate for real-time speech recognition and demonstrated the feasibility of achieving real-time recognition on the BBN Butterfly^{TM} Parallel Processor.

The speech recognition algorithm that is the focus of this effort is being developed for the BBN Byblos speech recognition system. The multiprocessor used in the research, the BBN Butterfly Parallel Processor, is a shared memory, MIMD machine. The algorithm was implemented using the Uniform System software methodology, a system that simplifies parallel programming without sacrificing efficiency. The algorithm is described, highlighting those portions critical to an efficient parallel implementation. Some of the problems encountered in trying to improve the efficiency are presented as well as the solutions to those problems. The algorithm is shown to achieve 79% processor utilization on a 97-node Butterfly Parallel Processor. This is equivalent to a speedup by a factor of 77 over a single processor benchmark.

We are beginning the second phase (Item 0002) of the effort, during which we will develop and demonstrate a real-time speech recognition capability within the Strategic Computing Program (SCP) Naval Battle Management application domain. We are extending and refining the parallel processing techniques developed in the first phase of this effort. These techniques will be applied to the implementation of a system for real-time large vocabulary continuous speech recognition. The vocabulary and grammar to be implemented and demonstrated will be a subset of the FRESH/CASES expert systems. The real-time system will be implemented on a BBN Butterfly Parallel Processor.

2. Efficient Implementation of Continuous Speech Recognition on a Large Scale Parallel Processor

The body of this section is a paper that will be presented at the IEEE International Conference on Acoustics, Speech, and Signal Processing, to be held in Dallas, TX on April 6-9, 1987. This paper describes the major technical achievements of this project in a concise format that is most appropriate for dissemination of the results to technical personnel of the Government, contractors, and other research laboratories. When published, the citation will be:

Kimball, Owen, Lynn Cosell, Richard Schwartz, and Michael Krasner, "Efficient Implementation of Continuous Speech Recognition on a Large Scale Parallel Processor", *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, April 6-9, 1987, Dallas, TX.

Efficient Implementation of Continuous Speech Recognition on a Large Scale Parallel Processor

Owen Kimball, Lynn Cosell,
Richard Schwartz, Michael Krasner

BBN Laboratories
10 Moulton St.
Cambridge, MA 02238

Abstract

This paper presents research into the use of large-scale parallelism for a continuous speech recognition algorithm. The algorithm, developed for the BBN Byblos system [1], uses context dependent Hidden-Markov models to achieve high recognition accuracy. The multiprocessor used in the research, the BBN Butterfly^{TM} Parallel Processor, is a shared memory, MIMD machine. The algorithm was implemented using the Uniform System software methodology, a system that simplifies parallel programming without sacrificing efficiency. The algorithm is described, highlighting those portions critical to an efficient parallel implementation. Some of the problems encountered in trying to improve efficiency are presented as well as the solutions to those problems. The algorithm is shown to achieve 79% processor utilization on a 97-node Butterfly Parallel Processor. This is equivalent to a speedup by a factor of 77 over a single processor benchmark.[1]

1. Introduction

The introduction of large-scale parallelism in computers offers the potential for greatly increased speed and better performance-cost ratios for algorithms that can make use of this parallelism. This paper describes the parallel implementation of a continuous-speech recognition algorithm that successfully uses the speedup provided by a general purpose multiprocessor, the Butterfly Parallel Processor.

The outline of this paper is as follows: Section 2 describes the Butterfly Parallel Processor and the Uniform System, Section 3 describes the BBN word recognition algorithm, Section 4 explains the initial parallel implementation of the algorithm, Section 5 describes the improvements to the algorithm for better processor utilization and presents results based on these improvements. The final section presents some conclusions from the work.

2. Butterfly and Uniform System

The Butterfly Parallel Processor [2] is composed of multiple (up to 256) identical nodes, each containing a processor and memory, interconnected by a high-performance switch. The Butterfly architecture is multiple-instruction-multiple-data-stream (MIMD), in which each processor node executes its own sequence of instructions, referencing data as specified by the instructions. Each processor node contains either a Motorola MC68000 or MC68020 microprocessor, an optional floating-point co-processor, from 1 to 4 megabytes of main memory, a co-processor called the Processor Node Controller, memory management hardware, an I-O bus, and an interface to the Butterfly switch.

The Butterfly switch allows each processor to access the memory on every other node. Collectively, these memories form the shared memory of the machine, a single address space accessible to every processor. All interprocessor communication is performed using shared memory. Instructions accessing memory on the same node as a processor typically take about 2 microseconds to complete, whereas those accessing memory on another node take about 5 or 6 microseconds. Block transfers from one memory to another run at 4 megabytes per second. The machines used in this project were 16-processor and 97-processor machines, each with 1 megabyte of memory and a MC68000 microprocessor on the processor nodes. Neither had hardware support for floating point arithmetic.

The software for the project was written using the Uniform System, a programming methodology supported by a library of high-level subroutines [3]. The benefit of using the Uniform System is that it can provide a simple, efficient solution to the problem of load balancing for the memory as well as for the processors. To balance the load on memory, the Uniform System routines spread out the data evenly across the different physical memories in the machine. Under the assumption that distributed data will also distribute memory accesses fairly evenly, this approach can reduce the inefficiency that results when many processors attempt to access the same memory simultaneously.

To balance the load on processors, the Uniform System treats processors as a pool of identical workers, all of which can execute the same tasks. In this way, tasks can be dynamically assigned to the free processors in the machine. In a typical program, control starts out in a single processor of the machine. To perform tasks in parallel, this processor calls a Uniform System "generator" subroutine, specifying a set of tasks to do and a task subroutine. The generator creates a descriptor of the work to be done and starts all processors. The processors then perform the work in parallel, each taking the next task data from the descriptor and executing the task routine with this data until all the work is completed. At that point, control is returned to the original single processor. An

[1]This work was sponsored by the Defense Advanced Research Projects Agency and was monitored by the Space and Naval Warfare Systems Command under Contract No. N00039-85-C-0313.

The authors would like to thank William Crowther for his assistance on the binary-tree maximum and for other informative discussions.

example of a simple generator is GenOnI. The call "GenOnI(task_routine, Ntasks)" assigns processors to perform the subroutine "task_routine" for every integer value in the range 1 to Ntasks.

3. Recognition Algorithm

The Byblos system has two major components, a trainer and a recognizer. The recognition component was implemented on the Butterfly Parallel Processor. The training component uses the forward-backward algorithm [4] to estimate discrete-density Hidden-Markov models of context-dependent phonemes. It combines these models to form word models that are used in recognition. The context-dependent models lead to accurate and robust recognition performance; the system has achieved 90% correct recognition on a 335 word speaker-dependent task with no grammar [5].

In the recognition process, input speech is analyzed every 10 ms and then vector quantized with a 256-vector codebook. The analysis and quantization are done in real time on an FPS array processor attached to a VAX. The quantization codes, each representing a frame of input speech, are input in real time over an ethernet connection to the search algorithm on the Butterfly Parallel Processor

The search algorithm finds the best scoring sequence of words using the trained word models. Each possible sequence of words that is considered is called a word sequence theory. The search uses the Viterbi decoding algorithm to update scores for all word sequence theories at each frame. In order to prevent underflow during score updating, all theory scores are normalized. To determine the "normalization factor" for a frame, the algorithm computes the maximum score of all states in all words in the frame and sets the factor to the score ceiling minus the maximum score.

The major work being performed in the algorithm can be abstracted in pseudo-code as follows:

```
FOR all input frames {
  max_score := 0
  best_end_score := 0
  FOR all words {
    update word score
    IF word_max_score > max_score
      max_score := word_max_score
    IF word_end_score > best_end_score
      best_end_score := word_end_score
  }
  determine initial state score for new theories from best_end_score
  determine normalizaton from max_score
}
determine and report best scoring theory
```

The algorithm computes two maxima: "max_score", the maximum over all states of the words scored in an input frame, and "best_end_score", the maximum score of all words' *final* states. The first maximum is used for the normalization factor mentioned above, and the second is used to determine the score for the initial state of all words in the next frame.

The core of this computation, the word score update, entails updating all the phonemes in a word. Each phoneme update requires a little less than one millisecond of computation, and the average word update time is slightly more than 4 milliseconds for the vocabularies used in this work.

4. Initial Parallel Implementation

As the first step toward a parallel implementation, the speech recognition program was ported from VAX/VMS to a single processor of the Butterfly Parallel Processor. Both versions of the program were in the language 'C'. The most significant change to the program in this phase was the use of the Uniform System memory management routines to store in global shared memory about 1.5 megabytes of data that had been stored on disk in the VAX version.

The VAX (and the first Butterfly System implementation) used floating-point arithmetic, but because floating-point arithmetic is performed in software in our Butterfly Parallel Processor, it is substantially slower than fixed point. For this reason, the program was switched to fixed-point arithmetic. As part of this change, multiplication of probabilities in the original version was converted to addition of corresponding log probabilities. With this modification, the execution time was about two minutes for a 3.5 second utterance from a 120 word task. This is about the same speed as our optimized floating point VAX 11/780 program.

Examination of the pseudo-code in the preceding section leads to a natural decomposition of the algorithm: the fundamental parallel task is to update the score of a single word for a single input frame. Using the Uniform system generator GenOnI, the pseudo-code for the parallel version of our algorithm becomes:

```
FOR all frames {
  best_end_score := 0
  max_score := 0
  GenOnI(update_word, N_words)
  determine initial state score for new theories from best_end_score
  determine normalizaton from max_score
}
determine and report best scoring theory
```

In this version, the subroutine update_word now includes the calculation of max_score and best_end_score. Note that since the processor calling GenOnI waits for all processors to finish before proceeding, this mechanism provides a synchronization that is needed to ensure that no processor begins updating words of a new frame until the initial state score and the normalization factor for the next frame have been computed.

Using this simple approach to parallel implementation, a first timing experiment was conducted using a 16-processor machine and a 120-word vocabulary task. Processor utilization was found to be 75%, i.e. the machine was effectively using the computation corresponding to 12 of the 16 actual processors. [6]. This result was judged to be good enough to proceed directly to work on a larger machine. The first time the program was run on a 97-processor machine, processor utilization was approximately 20%. Although this represents a factor of 20 speedup of the program, it is an inefficient use of the machine. The next section presents several factors that contributed to the inefficiency as well as the methods used to improve them.

5. Efficiency Improvements and Results

There are a number of potential obstacles to attaining efficient processor utilization on a multiprocessor. Typical issues include contention for a common memory location, serial code in the program, and processors waiting idly to synchronize with other processors. Each of the specific problems described below includes one or more of these issues.

Number of Tasks and Startup Overhead

Even before the program was run on a larger machine, we had anticipated that it would be hard to obtain high processor utilization with a vocabulary as small as 120 words. Since our long-term goal is to recognize speech from large vocabularies, we switched to a larger task of 335 words. This change improved processor utilization to 35% on the 97-processor machine.

The speed of processor scheduling was examined next. In the initial parallel version shown above, the generator subroutine call starts all the processors at each frame. It was found that the overhead of starting up was relatively large for the amount of work being done at each frame. To reduce the overhead, the program was altered to start all processors only once at utterance start, generating *Nframes*×*NWords* tasks at that point and letting each processor determine its word and frame indices from the single task index it receives from the generator. Processor utilization improved to about 50% with this change.

Processor Synchronization Issues

The task generation change had removed the synchronization provided by starting up a new generator at each frame. To replace this, an explicit synchronization was built into the program to be performed after all the words in a frame were processed. There were two subsequent changes to improve the efficiency of synchronization. The first dealt with task ordering. In the early versions of the algorithm, processors updated all the words in the vocabulary, with no particular ordering of the words. Since words have varying numbers of phonemes (from one to 14 phonemes in this task's vocabulary), different words took different amounts of time to update. If a processor began work on a long word near the end of the work for a frame, other processors would finish their assigned words and wait idly to synchronize with the one busy processor. To reduce this inefficiency, the words were processed in order from longest to shortest (in number of phonemes).

In figure 1, we schematically depict the situation before and after the words are ordered. The filled rectangles represent time when processors actively work on tasks and the white space represents time between tasks when no work is being accomplished. In the right hand part of the figure, idle processor time is substantially reduced by sorting.

The second change to synchronization efficiency concerned the point in the program at which synchronization was done. As mentioned, the purpose of the synchronization was to ensure that no processor proceeded to the next frame until the starting score for words and the normalization factor were computed. Since the normalization factor was only to avoid score underflow, it could be estimated a frame or more earlier. The only remaining synchronization constraint was the word-starting score. This score, however, is used only at the beginning of the *first* phoneme of each word. Considering this, the order of the update of a word was reversed so that the last phoneme was updated first, and the first phoneme updated last.

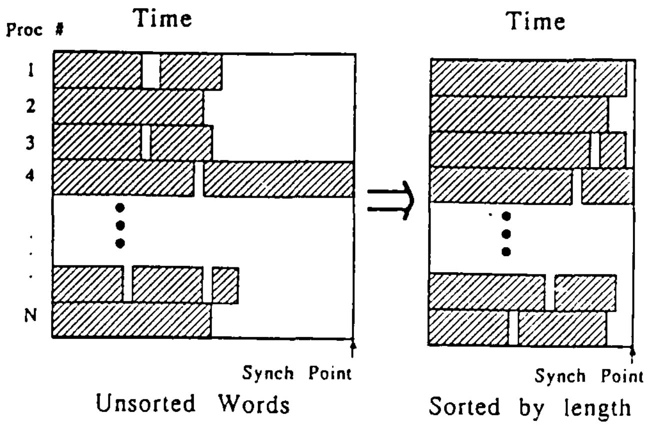


Figure 1: Ordering Tasks by Length

This change allowed a processor to finish work on one frame and immediately begin work on updating a word from the next frame, synchronizing only when it got to the first phoneme. In this way, time that had been previously spent by processors waiting for others to finish a frame was now being used to perform useful work from the next frame.

Figure 2 depicts the situation for two frames of an utterance before and after this change. Tasks for time T+1 are shown in two shades. The darker portion represents the part of the task that depends on the previous frame's work being finished. On the right, with the the order of the computation reversed, the idle processor time is reduced. The effect of the synchronization changes was to increase processor utilization to approximately 72%.

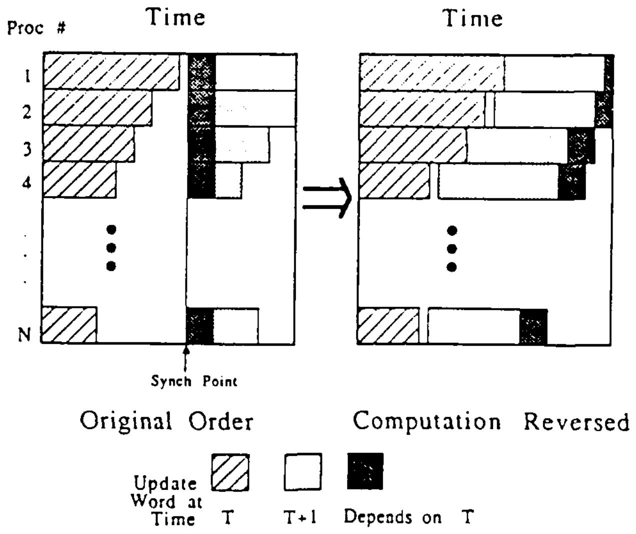


Figure 2: Reversing Word Computation Order

Finding Global Maximum

Finally, the efficiency of finding the maximum value was also improved. A straightforward computation of the maximum value requires that all values be compared with a single memory location, but this approach results in contention for that location. As a first improvement, the program was altered to make each processor maintain its own *local* maximum of the scores of all the words that it updates in a frame. At the end of the frame, the global maximum of these values over all processors was determined. In initial versions, this was accomplished by having processors sequentially compare their value to the global location and replace it if necessary. Although on a sixteen processor machine, the time for processors to turn in values in this way is negligible, with 97 processors, the inefficiency of the approach becomes noticeable.

An alternative to this approach was to set up a "binary tree" of locations for taking the maximum. In this approach, the processors' local maxima are the leaves of the tree and the maxima are propagated up through the nodes of the tree. This approach reduces the asymptotic time for finding a global maximum from O(N) to O(log N), where N is the number of processors. More importantly in our case, efficiency improved because memory contention was reduced.

The total effect of all the improvements described above was to improve processor utilization on a 97-processor machine from 20% to 79%. Figure 3 is a graph of processor utilization for 1 to 97 processors on the 335 word task. The actual speed of the speech recognition improved accordingly. After the optimizations are included, a one-processor Butterfly Parallel Processor requires 128 times real time (128 seconds to process one second of input speech) and a 97-processor machine requires about 1.7 times real time.

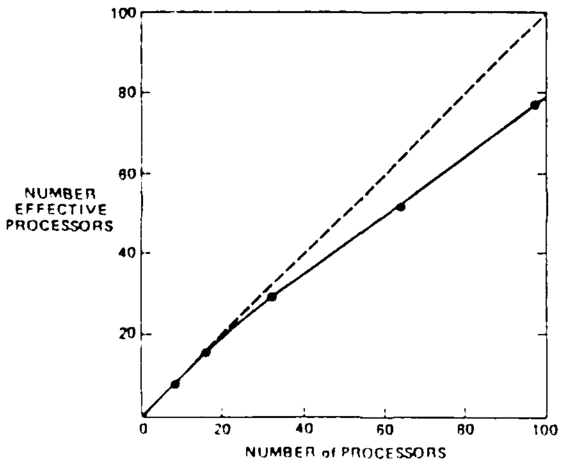


Figure 3: Butterfly Processor Utilization, 335 words

6. Conclusions

This work has shown that the Butterfly architecture is suitable for continuous speech word recognition. The algorithm was implemented efficiently without changing the type or amount of computation performed. Some ingenuity was required to obtain an efficient realization, but once the obstacles were understood, solutions presented themselves fairly readily. The memory and processor management functions of the Uniform System made initial parallelization of the algorithm quite easy and provided several alternatives for improving implementation efficiency when required.

We draw several broad conclusions about efficient parallel programming as well. Most obviously, and perhaps most importantly, it is crucial that sequentially executed code be eliminated wherever possible. Similarly, much of the inefficiency in our original multiprocessor program was due to processors waiting for each other. Synchronizing processors only after all other possible work is done was found to be a good strategy to avoid this. Additionally, it can be very important to minimize the overhead of parallel constructs such as starting processors.

References

1. Y.L. Chow, M.O. Dunham, O.A. Kimball, M.A. Krasner, G.F. Kubala, J. Makhoul, P.J. Price, S. Roucos, and R.M. Schwartz, "BYBLOS: The BBN Continuous Speech Recognition System", *ICASSP*, Dallas, TX, April 1987, Elsewhere in these proceedings

2. R. Thomas, R. Gurwitz, J. Goodhue, and D. Allen, "Butterfly Parallel Processor Overview", Tech. report, BBN, March 1986.

3. R. Thomas, "The Uniform System Approach to Programming the Butterfly Parallel Processor", Report 6149, BBN, June 1986.

4. L.E. Baum and J.A. Eagon, "An Inequality with Applications to Statistical Estimation for Probabilistic Functions of Markov Processes and to a Model of Ecology", *Amer. Math Soc. Bulletin*, Vol. 73, 1967, pp. 360-362.

5. Y.L. Chow, R.M. Schwartz, S. Roucos, O.A. Kimball, P.J. Price, G.F. Kubala, M.O. Dunham, M.A. Krasner, and J. Makhoul, "The Role of Word-Dependent Coarticulatory Effects in a Phoneme-Based Speech Recognition System", *ICASSP*, Tokyo, Japan, April 1986, pp. 1593-1596, Paper No. 30.9.1.

6. L.K. Cosell, O.A. Kimball, R.M. Schwartz, M.A. Krasner, "Continuous Speech Recognition on a Butterfly Parallel Processor", *Proceedings of the International Conference on Parallel Processing*, St. Charles, Illinois, August 1986, pp. 717-720.

3. Publications

The following publications were published as a result of this sponsored research:

1. Kimball, Owen, Lynn Cosell, Richard Schwartz, and Michael Krasner, "Efficient Implementation of Continuous Speech Recognition on a Large Scale Parallel Processor", *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, April 6-9, 1987, Dallas, TX.

2. Cosell, Lynn, Owen Kimball, Richard Schwartz, and Michael Krasner, "Continuous Speech Recognition on the Butterfly^{TM} Parallel Processor", *Proceedings of the International Conference on Parallel Processing*, August 19-22, 1986, St. Charles, IL.

3. Cosell, Lynn, Owen Kimball, Richard Schwartz, and Michael Krasner, "Implementation of Continuous Speech Recognition on a Butterfly^{TM} Parallel Processor", *Proceedings of the DARPA Strategic Computing Speech Recognition Workshop*, Science Applications International Corp. Report No. SAIC-86/1546, February 19-20, 1986, Palo Alto, CA, pp. 60-66.

4. Krasner, Michael, "A Parallel Processing Speech Recognition Architecture on the BBN Butterfly Multiprocessor", *Proceedings of the National Radio Science Meeting*, National Academies of Science and Engineering, January 13-16, 1986, Boulder, CO, p. 139.

END
2-87
DTIC