BEAM: An Accelerator for Speech Recognition

R. Bisiani, T. Anantharaman and L. Butcher

January 1989
CMU-CS-89-102

Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213

This is a revised version of a paper that will appear in the Proceeding of the IEEE 1989 International Conference on Acoustics, Speech and Signal Processing. This research is sponsored by the Defense Advanced Research Projects Agency, DoD, through ARPA Order 5167, and monitored by the Space and Naval Warfare Systems Command under contract N00039-85-C-0163. Views and conclusions contained in this document are those of the authors and should not be interpreted as representing official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or of the United States Government.

Abstract

BEAM is a hardware accelerator that has been designed and built for real-time execution of the SPHINX speaker-independent, continuous speech recognition system and similar systems. SPHINX on BEAM is able to recognize sentences from a 1,000 word vocabulary and a perplexity 60 grammar in about 1.3 times real time. BEAM does not use any custom integrated circuit. This report describes the architecture of the accelerator, gives performance data and compares BEAM to other architectures.

1. Introduction

BEAM is a multiprocessor capable of executing hidden-Markov-model (HMM) based speech recognition systems in real-time. BEAM has been applied to the SPHINX speech recognition system [1] and is currently the fastest machine able to run this recognition system. SPHINX is a speaker-independent, connected recognition system capable of high accuracy on non-trivial tasks. For example, it has less than 5% recognition error on a 1,000-word, perplexity 60 task (DARPA Resource Management Task, see [1] for a description of this task); when run on BEAM, SPHINX can recognize a sentence from this task in about 1.3 times real time. By comparison, a Sun-4/260 workstation is six times slower than BEAM on the same task and a 15-processor Encore Multimax shared-memory multiprocessor is about four times slower. This speed is achieved without compromising the accuracy of SPHINX.

Despite its speed, BEAM does not use any custom integrated circuits and has been developed quickly and cheaply (in about five months by three people). Unlike some custom integrated circuit systems, BEAM is programmable in a simple language (C) and can be tailored to different recognition systems. The main reason for the good performance of BEAM is that it is a good trade-off between general purpose systems, which do not have the necessary speed, and custom systems, which take too much time to build. We will first briefly describe the recognition task, then describe the architecture and finally compare it with other systems.

2. The Task

Speech recognition systems require a certain amount of *front-end* processing for the transformation of the input signal into a form suitable for the recognition algorithm. Front-end processing is of no concern from the point of view of the computational power required, since general-purpose digital signal processors can easily perform this task in real time. For example, a Texas Instruments TMS 32030, programmed in C, could perform the front-end processing required by SPHINX (including controlling the data-acquisition device) in real time. For our experiments we have used a signal processing board that employs three TMS 32020's. This board generates a vector that represents the likelihood (log probability) that the current 10ms speech input was generated by each one of the reference sound classes. This vector is the input of BEAM.

The task of BEAM is to search the HMM representation and return the most likely sequence of words. Allophones are described with seven-state models and words are sequences of (in average) six allophones. Therefore, each word requires about 40 states. Legal sequences of words are represented by a grammar that can be either a finite state grammar or a statistical grammar. The *likelihood* of a state S at time *t* is a function of the current input vector and of the likelihoods, at time *t-1*, of the states preceding S

in the HMM. There are a few minor differences in the computation depending on whether a transition crosses a word boundary or not.

In a regular Viterbi algorithm all the states are updated at every frame: for the DARPA 1,000-word Resource Management Task this would require updating about 40,000 states every 10ms, or one state every 200ns. In order to lessen the computational load, BEAM uses a heuristic search technique, called beam search, in which a number of nearly optimal alternatives (the *beam*) are examined in parallel at every frame. Beam search is a heuristic technique because heuristic rules are used to discard nonpromising alternatives in order to keep the size of the beam as small as possible. Beam search reduces the number of states that need to be updated at every frame to about 4,000, or about 8,000 transitions since there are, in average, two predecessors for each state. This means that beam search makes the task one order of magnitude less demanding, a factor that cannot be ignored. On the other hand, the size of the beam can vary dramatically and in an unpredictable way from one frame to the next: this creates load-balancing problems when decomposing a beam search algorithm for a multiprocessor. BEAM is equipped with special hardware to assist synchronization and load-balancing.

Memory bandwidth is probably the major bottleneck created by this algorithm. The DARPA task, for example, requires an average memory bandwidth of 40 Mbytes per second if it is executed in real time using a beam search algorithm (a regular Viterbi algorithm would require about ten times as much memory bandwidth). BEAM has a dual-bank shared memory and local memories to support memory bandwidth requirements.

3. The Architecture

The architecture employs only off-the-shelf components but it connects them in a way that improves their performance for this task, see Figure 3-1. Three general purpose Weitek 8032 processors share an 8-Mbyte memory and have two private memories: a 32-Kbyte program memory (8K instructions) and a 256-Kbyte local memory. Each of the processors is able to execute 10 million instructions per second, access the local memory at every instruction and access the shared memory every 200ns if there is no contention. Since there are two (low-order-bit interleaved) banks of shared memory on separate buses, two processors can access shared data at the same time.

The shared memory is augmented with a *one-bit-per-word* flag which is managed directly by the hardware and is visible at the instruction level through *special-read* and *special-write* operations. See the description of the HEP multiprocessor [2] for an example of the use of this feature in a general purpose machine. If the programmer accesses memory through special-read and write operations, the hardware checks the value of the flag before performing the operation: if the flag is 0, read operations are forbidden and if the flag is 1, write operations are forbidden. A forbidden operation leaves the memory untouched and sets a processor condition code. A successful special read sets the flag to 0 and a successful special write sets it to 1. In this way a producer-consumer synchronization operation between two processors can be implemented with a minimum of overhead.

The machine communicates with the rest of the world through a VME-bus connection that makes the shared memory available to a host. The flag-based synchronization is also available to the host.

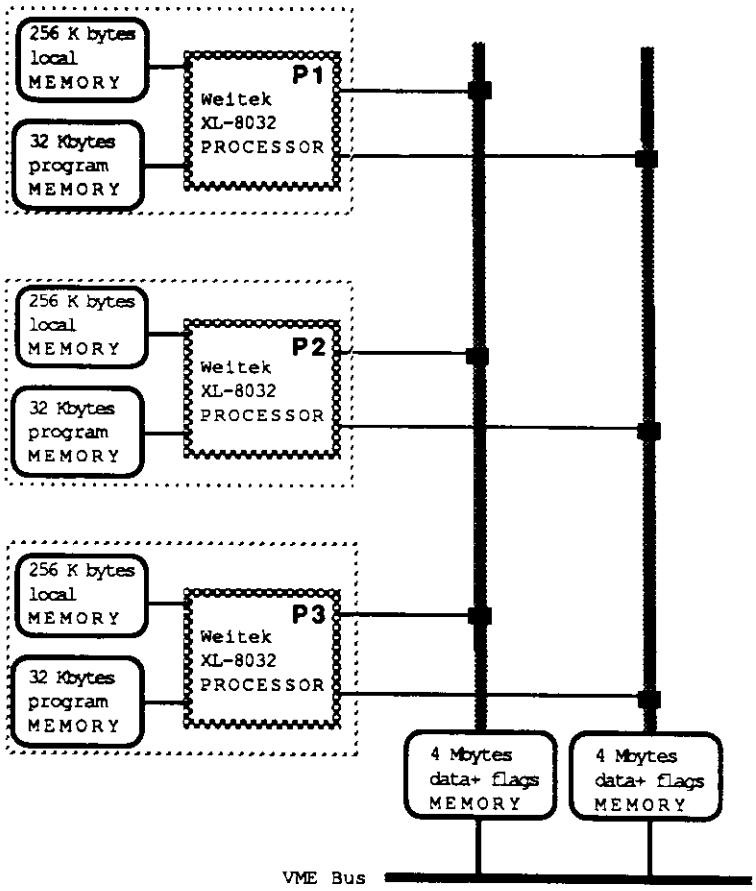

Figure 3-1: The Architecture of BEAM

Downloading of programs is done through the VME bus when the machine is not running.

4. Parallel Decomposition and Programming

Beam search requires a very fine-grain decomposition and a substantial amount of synchronization. Typically, see Figure 4-1, a processor must dequeue one state from the list of states to be processed, lock all the states that follow it and finally queue a new state. For the DARPA task to be executed in real time all these operations must be performed in about 2 μs. All intra-word transitions must be computed before inter-word computations begin, and vice-versa; this requires a barrier synchronization.

Barrier synchronization and locking of states can be accomplished very effectively with one-bit-per-word flags since they make it possible to overlap most of the synchronization overhead with data accesses. On the contrary, queuing (dequeuing) states in (from) a central queue causes a serialization that the special hardware cannot avoid. The solution is to split the queue into local queues, one for each processor, so that queue accesses can proceed in parallel. In this case, because of the highly data-dependent behavior of beam search, it is not likely that all queues will become empty at the same time and some processors will remain idle creating a load imbalance. This load imbalance can be limited by means of a rebalancing mechanism in which the processors negotiate how many queue items each should have and then hand their surplus to the processors that have a deficit. This mechanism has been

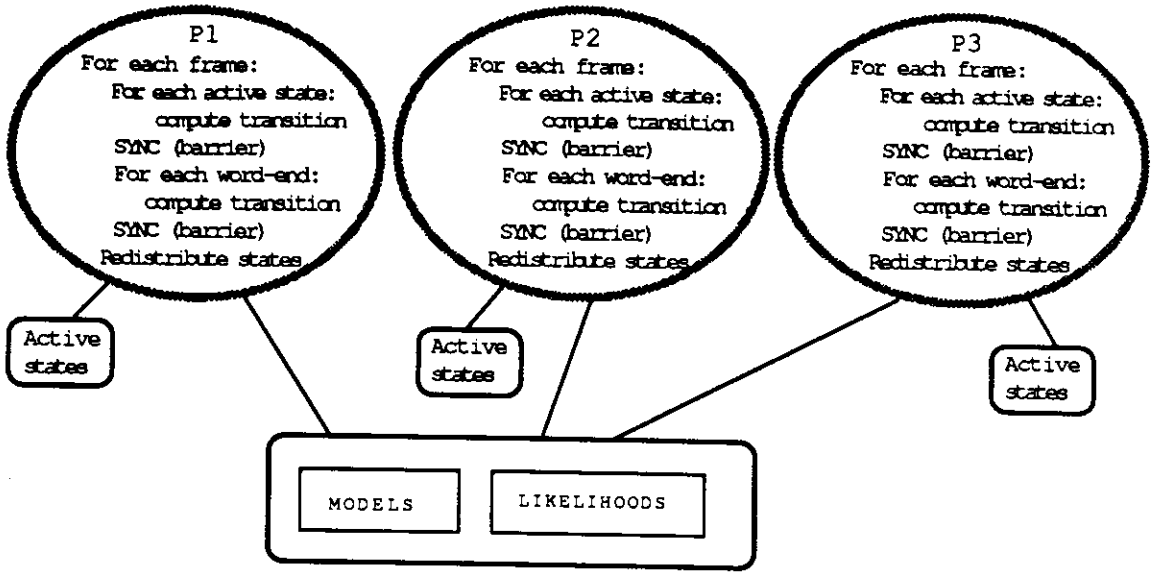

Figure 4-1: Parallel Decomposition

implemented very efficiently by using the synchronization flags.

The accelerator is programmed in C, this has let us optimize the implementation of the Viterbi search without compromising the possibility of improvements by the speech researchers. In fact, the accelerator has been built while the SPHINX system was still being developed. Most of the development time (80%) was spent writing software instead of designing or building hardware. This time was divided almost equally between support software coding, algorithm restructuring and algorithm coding. This seems to indicate that better software development methodologies will have a substantial impact on the development time of machines like BEAM.

5. Performance and Comparisons

The architectural requirements of speech recognition programs depend heavily on the task and on the recognition technique used. BEAM has been developed for HMM based systems with a vocabulary of 100 - 1,000 words. Table 5-1 shows the performance of BEAM on 150 test sentences of the 1000-word DARPA Resource Management Task. The performance is computed by dividing the recognition time (i.e. the time from the beginning the utterance to the moment the recognition is complete) by the length of the utterance. As one can see from the Table, the performance of a beam search algorithm depends on the sentence and on the speaker.

Perplexity	Average	St. Dev.	Min	Max
60	1.38	.40	.59	3.00
20	1.15	.33	.61	2.41

Figure 5-1: Recognition Time Divided by Utterance Length
(for Two Tasks of Different Perplexity)

The cost of an architecture suitable for HMM-based systems is mainly bound to the cost of memory (for example SPHINX needs about 8 Mbytes of memory). Most of the real estate and power requirements also depend on the amount of memory needed by the algorithm. Therefore, *for this task*, a custom-processor solution does not have any substantial advantage over BEAM. For example, a custom system that is being developed at SRI and UC Berkeley will eventually have the capability to process 50,000 states for every 10 ms frame. Since the system does not use pruning, it will probably be only slightly faster than BEAM on the 40,000 state DARPA Resource Management Task. Preliminary information indicate that the system will require five different custom integrated circuits and its board-level size will be comparable to BEAM's. The system has been in development for about one year.

Much simpler tasks, e.g. tasks with a vocabulary of less than 100 words or a highly constraining grammar, can be handled in real time by a single processor and a standard general-purpose architecture. Custom solutions are adequate in this case only if their cost is competitive with the cost of a general purpose processor, a difficult goal to meet. For example, a single Texas Instruments TMS 32030 could handle in real time a task with an average beam size of 1,000 states.

The feasibility of harder tasks, e.g. tasks with a ten times larger vocabulary or a very high perplexity grammar, has yet to be demonstrated. Therefore, we are unsure of the kind of speech technology necessary to achieve the required accuracy. For instance, it is not clear if continuous hidden-Markov models will be necessary. Recent history has shown that improving the computational capabilities of a speech recognition system does not necessarily make the system capable of tackling harder tasks. For example, there are architectures that can execute a Dynamic Time Warping algorithm in real time for vocabularies of many thousand words: unfortunately these architectures are not very useful because DTW algorithms do not work well on large vocabularies. There is no evidence that systems like SPHINX can be extended to larger and harder tasks and remain unchanged.

Only marginal speed-ups can be obtained on a general purpose machine. For instance, when ported to a 15-processor Encore Multimax, SPHINX is five times slower than on BEAM. There are two reasons for this. First, the memory bandwidth is exhausted with only 6 processors and the performance does not improve when more processors are used. Second, careful analysis has shown that, even if memory bandwidth could be increased until the memory bottleneck is eliminated, synchronization overhead would create a new bottleneck and about 50% of the time would be lost waiting for synchronization.

6. Conclusions

At the time BEAM was first demonstrated (June 1988) it was many times faster and cheaper than any other custom or general purpose system. Since then BEAM has been used daily to evaluate a number of speech recognition applications. Even though the performance of BEAM will be achievable by a single-processor, general purpose system when technology improves (the fate of all architectures), BEAM remains an example of how *general purpose technology* can be used to build systems that are substantially *faster* than *general purpose systems.*

Acknowledgments

Part of the BEAM support software has been developed by J.M. Calvez. K.F. Lee and H.W. Hon have helped us understanding SPHINX. D. Adams and R. Reddy have contributed to the project with their criticism and support.

References

[1] Lee,K.F.
Large-Vocabulary Speaker-Dependent Continuous Recognition: The SPHINX System.
PhD thesis, Carnegie-Mellon, 1988.

[2] Smith, B. J.
A Pipelined, Shared Resource MIMD Computer.
In *1978 Intl. Conf. on Parallel Processing*, pages 6-8. IEEE Computer Society, 1978.