Reconfigurable Computing for Speech Recognition: Preliminary Findings

S.J. Melnikoff, P.B. James-Roxby, S.F. Quigley & M.J. Russell

Presented at:
10th International Conference on Field Programmable Logic and Applications
(FPL 2000)
Villach, Austria, 28th-30th August 2000

Published in:
Lecture Notes in Computer Science #1896, pp.495-504
Springer-Verlag

Reconfigurable Computing for Speech Recognition: Preliminary Findings*

S.J. Melnikoff[1], P.B. James-Roxby[2], S.F. Quigley[1] & M.J. Russell[1]

[1] School of Electronic and Electrical Engineering, University of Birmingham, Edgbaston, Birmingham, B15 2TT, United Kingdom
S.J.Melnikoff@iee.org, S.F.Quigley@bham.ac.uk, M.J.Russell@bham.ac.uk
[2] Xilinx, Inc., 2300 55th Street, Boulder, Colorado 80301, USA
Phil.James-Roxby@xilinx.com

Abstract. Continuous real-time speech recognition is a highly computationally-demanding task, but one which can take good advantage of a parallel processing system. To this end, we describe proposals for, and preliminary findings of, research in implementing in programmable logic the decoder part of a speech recognition system. Recognition via Viterbi decoding of Hidden Markov Models is outlined, along with details of current implementations, which aim to exploit properties of the algorithm that could make it well-suited for devices such as FPGAs. The question of how to deal with limited resources, by reconfiguration or otherwise, is also addressed.

1 Introduction

Techniques for performing speech recognition have existed since the late 1960s, and since then, these have been implemented in both hardware and software.

A typical speech recognition system begins with a signal processing stage which converts the speech waveform into a sequence of acoustic feature vectors, or "observations". That data is then passed through a decoder which computes the sequence of words or phones (sub-word units, e.g. vowels and consonants) which is most likely to have given rise to the data. Higher-level information about context and grammar can be used to aid the process.

What is being proposed here is an implementation of the decoder. This is highly computationally demanding, but has the advantage that it is also highly parallelisable, and hence an ideal candidate for application in programmable logic.

With such devices - FPGAs in particular - becoming available which can utilise an increasing number of processing resources at ever faster speeds, this paper describes preliminary findings in research aimed at implementing speech recognition on a

* This research is funded by the UK Engineering and Physical Sciences Research Council

programmable logic device, while also looking at how to deal with the eventuality that the device used does not have sufficient logic resources to perform all of the necessary calculations at each step, by use of run-time reconfiguration or otherwise.

It is envisaged that such a decoder would act as a coprocessor in a larger system. This is advantageous both because it would free up resources for the rest of the system, and also because a dedicated speech processor is likely to be able to perform recognition faster than a general-purpose device.

This last point is particularly relevant here. At the time of writing, reconfigurable computing is still in its infancy, and so one of the aims of this research is to justify the use of this technology over conventional approaches (e.g. software implementations, ASICs, use of RAM, etc.).

The paper is organised as follows. Section 2 describes the theory of speech recognition based on the Hidden Markov Model. Section 3 then outlines some previous parallel implementations of this model, and describes a proposed structure for this implementation. This leads on to section 4, which discusses the findings of the current implementation, and looks at the problem of resource shortage. Section 5 describes the structure of the whole recognition system, and also looks at system performance. Section 6 then summarises the conclusions drawn so far, and section 7 outlines some of the tasks that will be carried out as the research continues.

2 Speech Recognition

The most widespread and successful approach to speech recognition is based on the Hidden Markov Model (HMM) [2], [8], [11], and is a probabilistic process which models spoken utterances as the outputs of finite state machines (FSMs).

2.1 The Speech Recognition Problem

The underlying problem is as follows. Given an observation sequence $O = O_1, O_2 \ldots O_T$, where each O_t is data representing speech which has been sampled at fixed intervals, and a number of potential models M, each of which is a representation of a particular spoken utterance (e.g. word or sub-word unit), we would like to find the model M which best describes the observation sequence, in the sense that the probability $P(M|O)$ is maximised (i.e. the probability that M is the best model given O).

This value cannot be found directly, but can be computed via Bayes' Theorem [11] by maximising $P(O|M)$. The resulting recognised utterance is the one represented by the model that is most likely to have produced O. The models themselves are based on HMMs.

2.2 The Hidden Markov Model

An *N*-state Markov Model is completely defined by a set of *N* states forming a finite state machine, and an $N \times N$ stochastic matrix defining transitions between states, whose elements a_{ij} = *P*(state *j* at time *t* | state *i* at time *t*-1); these are the *transition probabilities*.

With a Hidden Markov Model, each state additionally has associated with it a probability density function $b_j(O_t)$ which determines the probability that state *j* emits a particular observation O_t at time *t* (the model is "hidden" because any state could have emitted the current observation). The p.d.f. can be continuous or discrete; accordingly the pre-processed speech data can be a multi-dimensional vector or a single quantised value. $b_j(O_t)$ is known as the *observation probability*.

Such a model can only generate an observation sequence $O = O_1, O_2 \ldots O_T$ via a state sequence of length *T*, as a state only emits one observation at each time *t*. The set of all such state sequences can be represented as routes through the state-time trellis shown in Fig. 1. The $(j,t)^{\text{th}}$ node (a state within the trellis) corresponds to the hypothesis that observation O_t was generated by state *j*. Two nodes (*i*,*t*-1) and (*j*,*t*) are connected if and only if $a_{ij} > 0$.

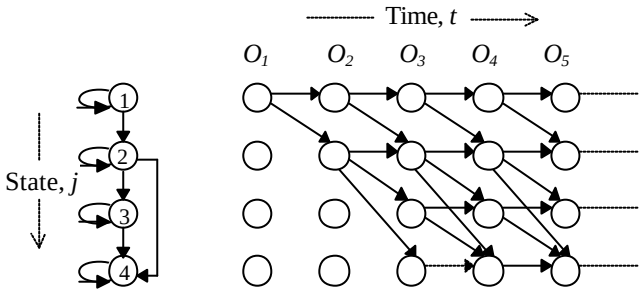

Fig. 1. Hidden Markov Model, showing the finite state machine for the HMM (*left*), the observation sequence (*top*), and all the possible routes through the trellis (*arrowed lines*)

As described above, we compute *P*(*M*|*O*) by first computing *P*(*O*|*M*). Given a state sequence $Q = q_1, q_2 \ldots q_T$, where the state at time *t* is q_t, the joint probability, given a model *M*, of state sequence *Q* and observation sequence *O* is given by:

$$P(O,Q|M) = b_1(O_1) \prod_{t=2}^{T} a_{q_{t-1}q_t} b_{q_t}(O_t), \tag{1}$$

assuming the HMM is in state 1 at time *t* = 1. *P*(*O*|*M*) is then the sum of all possible routes through the trellis, i.e.

$$P(O|M) = \sum_{\text{all } Q} P(O,Q|M). \tag{2}$$

2.3 Viterbi Decoding

In practice, the probability $P(O|M)$ is approximated by the probability associated with the state sequence which *maximises* $P(O,Q|M)$. This probability is computed efficiently using Viterbi decoding.

Firstly, we define the value $d_t(j)$, which is the maximum probability that the HMM is in state j at time t. It is equal to the probability of the most likely partial state sequence $q_1, q_2 \ldots q_t$, which emits observation sequence $O = O_1, O_2 \ldots O_t$, and which ends in state j:

$$d_t(j) = \max_{q_1, q_2 \ldots q_t} P(q_1, q_2 \ldots q_t; q_t = j; O_1, O_2 \ldots O_t | M). \quad (3)$$

It follows from equations (1) and (3) that the value of $d_t(j)$ can be computed recursively as follows:

$$d_t(j) = \max_{1 \le i \le N} [d_{t-1}(i) a_{ij}] \cdot b_j(O_t), \quad (4)$$

where i is the previous state (i.e. at time t-1).

This value determines the most likely predecessor state $y_t(j)$, for the current state j at time t, given by:

$$y_t(j) = \arg\max_{1 \le i \le N} [d_{t-1}(i) a_{ij}]. \quad (5)$$

Each utterance has an HMM representing it, and so the most likely state sequence not only describes the most likely route through a particular HMM, but by concatenation provides the most likely sequence of HMMs, and hence the most likely sequence of phones uttered.

3 Parallel Implementation

3.1 Previous Implementations

Parallel implementations of speech recognition systems have been produced before, most using HMMs. In contrast to the approach described here, previous implementations have generally used multiple processing elements (PEs) of varying sophistication, either at the board or ASIC level, rather than a programmable logic device.

Typically, the recognition problem has been broken down with each PE dealing with one HMM node. For example, [4] has an array of PEs that mirrors the structure of the trellis. One issue that has arisen with some previous parallel implementations is the problem of balancing the workload among a limited number of PEs, which results in a speedup that is less than linear. Steps can be taken to avoid redundant calculations (e.g. “pruning” paths whose probabilities fall below a threshold [9]), but this is more

difficult on parallel architectures than on serial ones [4]. Other approaches to parallel implementation include processor farms to automatically balance the load [1], [10], a more coarse-grained distributed computing model [3], [10], a tree-based architecture [9], or custom ICs with fine-grained PEs [7].

By using programmable logic, not only do we effectively have as many PEs as we want, but each PE can be optimised to handle the calculations for a single node. In addition, devices with (global) on-chip RAM are particularly suitable, as a buffer is needed to store the best predecessor states at each stage, for the purposes of backtracking.

Hence programmable logic, having not been applied to speech recognition in this way, has properties that may give it an edge over previous parallel implementations.

3.2 Proposed Structure

As described above, in order to perform Viterbi decoding, the trellis must be traversed forwards to find the best path, then once the observation sequence has ended, backtracking takes place, during which the best path is traced in reverse in order to extract the state sequence taken.

Forward Computation. Each node in the trellis must evaluate equations (4) and (5). This consists of multiplying each predecessor node's probability $d_{t-1}(i)$ by the transition probability a_{ij}, and comparing all of these values. The most likely is multiplied by the observation probability $b_j(O_t)$ to produce the result.

After a number of stages of multiplying probabilities in this way, the result is likely to be very small. In addition, without some scaling method, it demands a large dynamic range of floating point numbers, and implementing floating point multiplication requires more resources than for fixed point.

A convenient alternative is therefore to perform all calculations in the log domain. This converts all multiplications to additions, and narrows the dynamic range, thereby reducing all the arithmetic to (ideally) fixed point additions and comparisons, without in any way affecting the validity of the results obtained. Hence equation (4) becomes

$$d_t(j) = \max_{1 \le i \le N} [d_{t-1}(i) + \log a_{ij}] + \log[b_j(O_t)]. \quad (6)$$

The result of these changes mean that a node can have the structure shown in Fig. 2. The figure highlights the fact that each node is dependent only on the outputs of nodes at time $t-1$, hence all nodes in all HMMs at time t can perform their calculations in parallel.

The way in which this can be implemented is to deal with an entire column of nodes of the trellis in parallel. As the speech data comes in as a stream, we can only deal with one observation vector at a time, and so we only need to implement one column of the trellis. The new data values (observation vector O_t and maximal path probabilities $d_{t-1}(j)$) pass through the column, and the resulting d_t values are latched, ready to be used as the new inputs to the column when the next observation data appears.

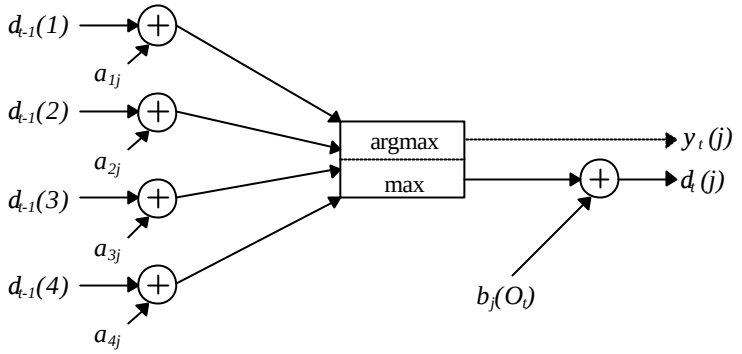

Fig. 2. Block diagram of a node representing state j in a 4-state finite state machine, with all calculations performed in the log domain. $\delta_{t-1}(i)$ are the outputs of previous nodes; O_t is the current observation vector. The transition probabilities a_{ij} and the observation probability distribution $b_j(O_t)$ are fixed for a specific node

Backtracking. Each node outputs its most likely predecessor state $y_t(j)$, which is stored in a sequential buffer external to the nodes. When the current observation sequence reaches its end at time T, a sequencer module reads the most likely final state from the buffer, chosen according to the highest value of $\delta_t(j)$. It then uses this as a pointer to the collection of penultimate states to find the most likely state at time T-1, and continues with backtracking in this way until the start of the buffer is reached. In the event that the backtracking buffer is filled before the observation sequence ends, techniques exist for finding the maximal or near-maximal path.

As the resulting state sequence will be produced in reverse, it is stored in a sequencer until the backtracking is complete, before being output. This state sequence reveals which HMMs have been traversed, and hence which words or sub-word units have been uttered. This information can then be passed to software which assembles the utterances back into words and sentences.

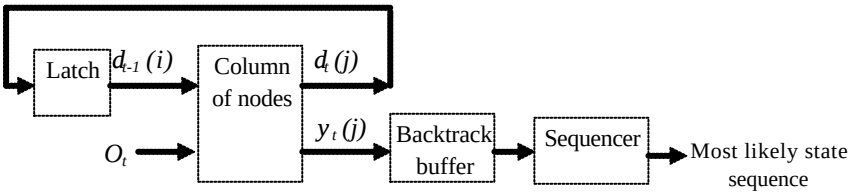

Fig. 3. Decoder structure. The nodes' outputs from time t–1 are supplied as their inputs at time t, along with the new observation vector O_t. The most likely predecessors of each state $y_t(j)$ are stored in the backtrack buffer until the speech data ends, then sent to the sequencer which traces the most likely path in reverse, before outputting it in the correct order

The structure of the decoder is shown in Fig. 3. Note that there will be additional logic in order to initialise the system, and to scale the values of δ_t in order to prevent overflow.

4 Implementation in Programmable Logic

4.1 Discrete HMM-Based Implementation

Parameters for a discrete HMM-based system have been obtained using speech recognition tools and real speech waveforms. In this case, each speech observation takes the form of an 8-bit value, corresponding to the address in a 256-entry table which describes the observation probability distribution $b_j(O_t)$ as a set of 15-bit vectors. Each node in the state-time trellis has a table associated with it. The transition probabilities are encoded in the same format.

We are using 49 HMMs, each one representing a monophone, i.e. an individual English vowel or consonant. Each HMM has 3 emitting states, so there are $3 \times 49 = 147$ nodes. Treating the speech observation data as an address, we need to obtain the value at this address for each node. If we do all this for each node in parallel, we need a 2205-bit-wide data bus.

The choice is therefore whether to store this data in on-chip RAM, use the LUTs as distributed RAM/ROM, or to store it off-chip.

Off-Chip RAM. If we were to store the data off-chip, we would have to perform several reads per observation vector, as the kind of data width required would not be realisable. In addition, off-chip RAM is likely to be slower than any on-chip alternative. However, dedicated RAM can provide more data storage than is available on an FPGA, which becomes particularly relevant as the recognition system is made more complex.

On-Chip RAM. On-chip RAM can offer increased speed, and very high data width. A trellis column containing 11 of the 49 HMMs has been implemented on a Xilinx Virtex XCV1000. It requires 31 of the 32 Block RAMs on the FPGA, plus around 4000 of the 24,500 LUTs (16%) for the addition and compare logic, and allows all 33 nodes to obtain their observation probabilities in parallel at an estimated clock frequency of 50MHz.

From these figures, we can predict that an XCV1000 could store around 70 HMMs. If more were required, we would have to use reconfiguration or off-chip RAM, and split the HMM column up, processing it in sections. At 50MHz, even allowing for a deep pipeline and access or reconfiguration delays, this would permit of the order of thousands of HMMs to be handled within the 10ms allowed for real-time speech processing.

While this gives a useful prediction of resource usage, clearly a larger FPGA is required for a full implementation, not least because the above figures do not include the resources needed for the backtracking buffer (which is likely to require on-board RAM as well), scaler, and other control logic.

Distributed RAM/ROM. LUTs can typically be configured as distributed RAM or ROM. While using these for storing the observation probabilities is likely to result in faster accesses than for Block RAM, it is at the expense of resources that need to be used for other parts of the system - whereas using Block RAM does not incur this penalty.

4.2 Reconfiguration vs. RAM [5]

A larger device may alleviate some of these problems, but speech recognition systems can easily be made more complex and so eat into any spare resources (explicit duration modelling [4], [6], [8] is one such resource-hungry improvement). It is therefore necessary to consider early on which of the above options should be used.

One possible solution is to use distributed ROM with run-time reconfiguration (RTR). Focussing on the observation probabilities, the only parts of the FPGA that need to be reconfigured are some of the LUTs; the associated control logic is identical for each node, and does not need to be changed (the transition probabilities would be different, but require significantly less data, and so could, in theory, be stored on-chip). In addition, the system control logic can remained fixed.

Given this situation, an FPGA which is too small to store all the required data could perhaps be repeatedly reconfigured at run-time by overlaying the LUTs holding the probabilities, so that each new observation vector is passed through all the HMMs in the system.

If on-chip RAM is too small, the only alternative is to perform a number of reads from off-chip RAM. A key deciding factor in whether to do this rather than RTR is the speed with which each can be performed. For RAM in particular, we are limited by the width of the data bus, which will obviously determine how many reads we need to do for each speech observation.

It remains to be seen which method will be the more suitable for this application.

5 Speech Recognition Systems

5.1 System Structure

At present, we are using an XCV1000-6 BG560, which resides on an ESL RC1000-PP prototyping board. The board is a PCI card which sits inside a host PC.

Recorded speech waveforms are pre-processed in software, and stored as quantised data. Once the initial system is completed, the speech data will be sent to the FPGA via the PCI bus; the FPGA will then perform the decoding and output a state sequence to the PC, which will map it back into phones.

In other words, the FPGA will be acting as a coprocessor, dealing with the most computationally-demanding part of the recognition process, thereby reducing the load on the PC's processor.

5.2 Performance

A significant and unique property of speech is that for the purposes of recognition, we can sample it at a mere 100Hz, giving us 10ms to process each piece of data - assuming that we are always aiming to do recognition in real-time. This means that whether we use RTR or off-chip RAM, there is a large period available to perform these operations, which provides a lot of flexibility when it comes to making the decoding algorithm more complex.

At this data rate, the pre-processing can be done in software in real-time. While this does not rule out eventually doing this on the FPGA as well, for the time being the FPGA will be used purely for the decoding.

6 Conclusion

So far, we have investigated Hidden Markov Model Viterbi decoding as a method of speech recognition, and have broken down the process in such as a way so as to take advantage of a parallel computing architecture.

Based on this analysis, we have begun work on an implementation of a real-time monophone speech decoder in programmable logic, which is expected to fit comfortably within an XCV1000, while utilising off-chip RAM. We believe that even if future FPGAs (or other similar devices) have the capacity to deal with a basic HMM-based algorithm, improvements can be made which, while making recognition more effective, would require off-chip RAM access or run-time reconfiguration in order to deal with the increase in processing resources needed.

7 Future Work

This research is clearly at an early stage, and so there are a number of issues which still need to be dealt with, including:

- **Integrate the recogniser into a complete system**: merely synthesising a recogniser will provide useful information on resource usage, but we need to able to test it! This will require using the FPGA prototyping board mentioned above, and writing software for the PC that houses it. It is envisaged that only recorded speech will be used for the time being. The results will then be tested against the output of an already completed software version of the recogniser.
- **Improve the recognition system**: once a monophone recogniser is completed, the next logical step is to move on to a bigram- and trigram-based system (pairs and triples of monophones). Whereas a monophone recogniser requires 49 HMMs, a bigram/trigram version uses around 500-600, another reason why being able to cope with limited resources is very important for this application.
- **Use of semi-continuous and continuous HMMs**: FPGAs are particularly well suited to dealing with discrete (quantised) speech data. However, use of continuous data

has been shown to produce better results in terms of recognition accuracy. Implementing this requires computing sums of Normal distributions (Gaussian mixtures) on an FPGA.

References

1. Alexandres, S., Moran, J., Carazo, J. & Santos, A., "Parallel architecture for real-time speech recognition in Spanish," *Proc. IEEE International Conference On Acoustics, Speech And Signal Processing (ICASSP '90)*, 1990, pp.977-980.
2. Cox, S.J., "Hidden Markov models for automatic speech recognition: theory and application," *British Telecom Technology Journal*, **6**, No.2, 1988, pp.105-115.
3. Kimball, O., Cosell, L., Schwarz, R. & Krasner, M., "Efficient implementation of continuous speech recognition on a large scale parallel processor," *Proc. IEEE International Conference On Acoustics, Speech And Signal Processing (ICASSP '87)*, 1987, pp.852-855.
4. Mitchell, C.D., Harper, M.P., Jamieson, L.H. & Helzerman, R.A., "A parallel implementation of a hidden Markov model with duration modeling for speech recognition," *Digital Signal Processing*, **5**, No.1, 1995, pp.43-57 *and* http://purcell.ecn.purdue.edu/~speechg/
5. James-Roxby, P.B. & Blodget, B., "Adapting constant multipliers in a neural network implementation," *to appear in: Proc. IEEE Symposium on FPGAs for Custom Computing Machines (FCCM 2000)*, 2000.
6. Levinson, S.E., "Continuously variable duration hidden Markov models for automatic speech recognition," *Computer Speech and Language*, 1986, **1**, pp.29-45.
7. Murveit, H. *et al*, "Large-vocabulary real-time continuous-speech recognition system," *Proc. IEEE International Conference On Acoustics, Speech And Signal Processing (ICASSP '89)*, 1989, pp.789-792.
8. Rabiner, L.R., "A tutorial on hidden Markov models and selected applications in speech recognition," *Proceedings of the IEEE*, **77**, No.2, 1989, pp.257-286 *and* Waible, A. & Lee, K.F. (eds.), *Readings in Speech Recognition*, 1990, Morgan Kaufmann Publishers, Inc., pp.267-296.
9. Roe, D.B., Gorin, A.L. & Ramesh, P. "Incorporating syntax into the level-building algorithm on a tree-structured parallel computer," *Proc. IEEE International Conference On Acoustics, Speech And Signal Processing (ICASSP '89)*, 1989, pp.778-781.
10. Sutherland, A.M., Campbell, M., Ariki, Y. & Jack, M.A., "OSPREY: a transputer based continuous speech recognition system," *Proc. IEEE International Conference On Acoustics, Speech And Signal Processing (ICASSP '90)*, 1990, pp.949-952.
11. Young, S., "A review of large-vocabulary continuous-speech recognition," *IEEE Signal Processing Magazine*, **13**, No.5, 1996, pp.45-57.