Hardware Acceleration of Hidden Markov Model Decoding for Person Detection

Suhaib A. Fahmy, Peter Y.K. Cheung
Department of Electrical and Electronic Engineering
Imperial College London
Exhibition Road
London, SW7 2BT, England
{s.fahmy, p.cheung}@imperial.ac.uk

Wayne Luk
Department of Computing
Imperial College London
180 Queen's Gate
London, SW7 2AZ, England
wl@doc.ic.ac.uk

Abstract

This paper explores methods for hardware acceleration of Hidden Markov Model (HMM) decoding for the detection of persons in still images. Our architecture exploits the inherent structure of the HMM trellis to optimise a Viterbi decoder for extracting the state sequence from observation features. Further performance enhancement is obtained by computing the HMM trellis states in parallel. The resulting hardware decoder architecture is mapped onto a field programmable gate array (FPGA). The performance and resource usage of our design is investigated for different levels of parallelism. Performance advantages over software are evaluated. We show how this work contributes to a real-time system for person-tracking in video-sequences.

1. Introduction

Use of the Hidden Markov Model for recognition has been gaining in popularity. The foremost application has been in speech recognition, with research going back nearly 20 years. In the computer vision domain, much activity has been seen recently, with the HMM being used for character recognition in deformed text [5], template matching [2] and face recognition [7]. The strength of the Hidden Markov Model is in its ability to cope with deformity to the image being recognised. This recognition ability of the HMM has been taken advantage of in a proposed person-tracking system that serves as the background for this work.

Unfortunately one of the main difficulties with the use of the Hidden Markov Model is its computational complexity. Implementation in hardware seems an ideal solution to this problem, in order to allow faster processing. While much work has been done with the Hidden Markov Model for speech recognition, as far as we are aware this paper is the first to explore a hardware architecture and implementation of the Hidden Markov Model specifically for vision systems.

It is important to note that the HMM as used in these systems uses offline learning. That is, the model is taught, until accurate parameters are obtained. These parameters are then used in recognition systems such as ours. The recognition system does not adapt to its input over time, although the reconfigurable nature of FPGAs means that we can change our system parameters by writing a completely new design to the chip.

Much of the work in this paper is inspired by a software design of a person-tracking system proposed by Rigoll et al. [10, 9, 3, 4] The focus of this paper is on the acceleration of the state decoding part of the system, as it is the most complex and suited to hardware. If the decoding could be accelerated sufficiently, the system could become feasible for real-time processing.

The specific contributions of this paper are: (a) exploitation of the inherent structure of the HMM trellis to simplify and parallelise Viterbi decoding; (b) mapping of the Viterbi decoding stage into reconfigurable hardware, and (c) the evaluation of its performance and resource usage for different levels of parallelism.

2. The Hidden Markov Model

The Hidden Markov Model is essentially an extension of a standard Markov-process state machine. The idea is that there exists a process which goes through a number of states. These states are not directly observable, but some other observation can be made that is statistically linked to the state of the process. By knowing the sequence of observations and the properties of the process, the state sequence can be deduced.

This is the "state decoding" problem of HMMs. The information available is as follows: $A = \{a_{ij}\}$ where $a_{ij} = Pr(q_j \text{ at } t | q_i \text{ at } t-1)$, the state-transition probabilities;

$B = \{b_j(O)\}$ where $b_j(O) = Pr(O \text{ at } t | q_j \text{ at } t)$, the observation probabilities and $\pi = \{\pi_i\}$ where π_i are the initial state probabilities and q_i are the states. [8]

Some important notes are that there is only one observation and state-transition per timestep, and the state-transition and observation probabilities do not change over time.

Recognition using HMMs relies on the Viterbi algorithm to extract the state sequence from a series of observations. The Viterbi algorithm has been widely researched and efficient implementations in the field of block-convolution decoding and speech-recognition have been proposed. [1, 12]

The state-decoding problem consists of solving the recursive equations in 1 and 2. $\delta_t(j)$ computes the probability of being in state j in timestep t, while $\psi_t(j)$ gives the most likely predecessor of state j at time t.

$$\delta_t(j) = \max_{0 \le i \le N} [\delta_{t-1}(i) \cdot a_{ij}] \cdot b_j(O_t) \tag{1}$$

$$\psi_t(j) = \arg\max_{0 \le i \le N} [\delta_{t-1}(i) \cdot a_{ij}] \tag{2}$$

The state sequence is obtained when δ and ψ are computed for the last timestep. The state with the greatest value of δ is taken to be the final state. The value of ψ for that state is then used to find the predecessor and the backtracking process continues recursively until a full state sequence has been obtained.

2.1. 2-Dimensional Representation

The state sequences discussed so far have all been one dimensional. However, for application of this theory to images and visual data, some way must be found to extend the model to be applicable in two dimensions. A fully connected model would have very computationally-intensive training and recognition algorithms, and so is avoided. Another method is to let the states in a one-dimensional HMM be themselves HMMs. This is called the embedded Hidden Markov Model. [5] The structure can be simplified further by flattening which gives a state-representation as shown in Figure 1.

It is important to note that this is not a true 2-dimensional representation, since transitions from column-to-column are not possible. This is called the Pseudo 2-Dimensional Hidden Markov Model. The state representation above is the one used in our system.

2.2. Application Considerations

The above simplifications mean that in our consideration of a design for decoding of the HMM state sequence, it is worth thinking from scratch. Much of the work on the optimisation of the Viterbi algorithm for HMM decoding is specific to the domain of speech recognition, and deals with more complex state sequences. The same can be said of work done on the Viterbi decoder for convolutional decoding. Specifically, in convolutional codes, the number of states is typically smaller, but the transitions are much more complex.

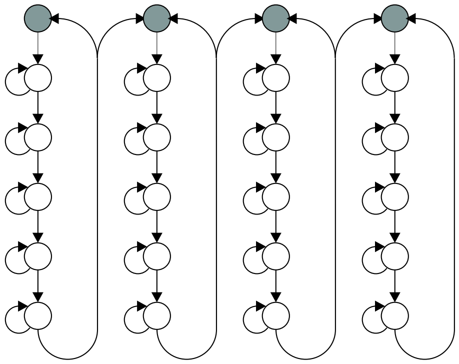

Figure 1. State representation of the HMM

2.3. Log Domain Representation

In computing the state-sequence, a multiplication is needed for each predecessor, and one more for multiplying by the observation probability, as seen in (1). Given the recursive nature of the equation, and these values being probabilities, dynamic range is an issue that must be tackled. One way of overcoming this issue is to perform these calculations in the log-domain. This reduces multiplications to additions and allows the wide dynamic range to be represented with fewer bits. Furthermore, since we are concerned with relative rather than exact values, the loss of precision is not of much importance.

Given that the log of a probability is always negative, we negate the result to do away with the need for dealing with signed arithmetic. [6] Therefore the max becomes a min. In the log domain, we now compute:

$$\delta_t(j) = \min_{0 \le i \le N-1} [\delta_{t-1}(i) + a_{ij}] + b_j(O_t) \tag{3}$$

$$\psi_t(j) = \arg\min_{0 \le i \le N} [\delta_{t-1}(i) + a_{ij}] \tag{4}$$

2.4. Trellis Structure

We have presented the general form of the Viterbi algorithm for deduction of a state-sequence from a series of observations. For each timestep, we must compute the probability of being in each state as defined in the equations. This calculation depends upon the probabilities of each of the states from the previous timestep and the observation probability for each state in the current timestep. Thus for a system with N states, and an observation sequence T timesteps long, the number of multiplications is $(N^2+1)\cdot T$. In our case, the number of states is 24 and the typical number of observations per image is in the region of 3000. It is clear that if the state-sequence is computed using the standard equations, it will be too computationally intensive for real-time processing at 30 frames/sec.

Looking at Figure 1, one can see that the state transitions in our system are not fully connected. The state transition trellis is shown in Figure 2. It is clear in fact that each state only has 2 predecessors. Taking advantage of this would simplify our calculation immensely, reducing the number of multiplications to $(2N+1)\cdot T$. That is a reduction in computation of around 90%.

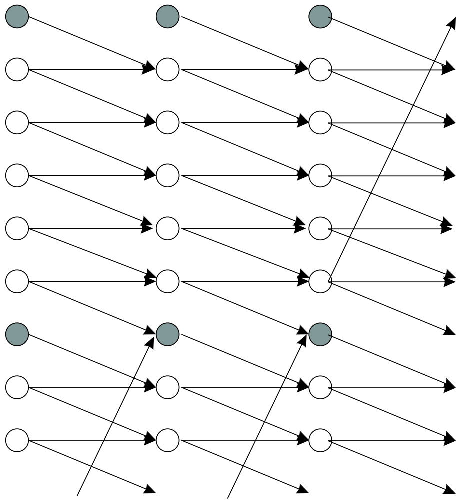

Figure 2. Extract from the state-transition trellis

It is also clear that the state transition sequence follows a fixed pattern. The general case is that 2 possible predecessors for each node N in timestep T are the nodes $N-1$ and N from timestep $T-1$. However, in the case of states 1,7,13 and 19 the predecessors are nodes $N-1$ and $N+5$ from the previous timestep. Hence, we can design an efficient node-calculation unit that has 2 inputs, one of which depends on which state we are computing the result for.

2.5. Inherent Parallelism

Another property that suggests hardware would be much more suited to HMM decoding than software is the inherent parallelism in the Trellis. In a normal software implementation, each node within a timestep is calculated in turn, before moving onto the next timestep. This means that the system can only cope with an observation rate that allows it to compute all nodes in the inter-observation time. In our case this means 24 calculation-times must complete before the arrival of the next observation.

From the trellis diagram, it is clear that nodes in one timestep only depend upon values in the previous timestep. This means that more than one node can be calculated in parallel. In fact, given sufficient resources, all nodes in one timestep could be calculated in parallel. This allows for a higher observation-rate in line with our aim of realtime processing.

We will investigate in the next section, the effects of different levels of parallelism on speed and area of our implementation.

3. Results

The basic idea of our design is to implement a "decoder node", that goes through each state to compute the δ and ψ values for the current timestep. The node is a simplified solver for (3) and (4), taking into account the simplifications mentioned in Section 2.4. A primitive example of the design is shown in Figure 3. The results from the calculations in the previous timestep are fed into the unit. For calculation of the result for state N, the only possible predecessors are states $N-1$, N and $N+5$. These are fed as inputs along with the appropriate state transition probabilities and the observation probability for the current state. A *select* signal goes high when we are computing the results for states 1,7,13 and 19. This causes the values for state $N+5$ from the previous timestep to be used instead of state N. The comparator chooses the minimum of the two values and stores the most-likely predecessor. The observation probability is then added to give the final result for this timestep.

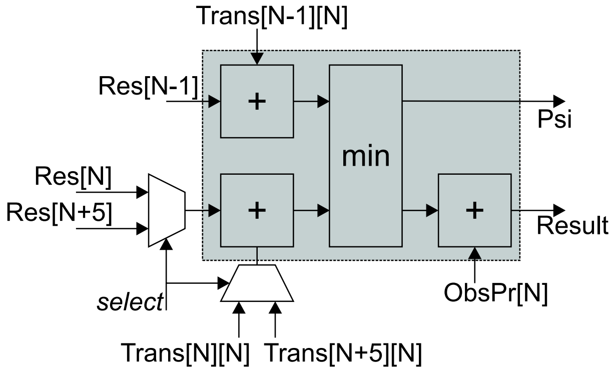

Figure 3. The efficient node design

3.1. Unit Context

It is important to consider where this unit fits in to the rest of our system. As proposed by Rigoll et al [10, 9, 3, 4], the Hidden Markov Model in our system is used to identify the presence of a person. First comes the person-detection phase: through background subtraction, the moving object is extracted. A bounding-box is formed with the addition of some margin on each side. This image segment is then processed with a block based on the Discrete Cosine Transform, using an overlapping sliding window, to extract features. These are presented to the pre-trained HMM block as observations, and the block decides whether or not a person is present, by taking into account the number of person states in the extracted sequence.

Once the presence of a person has been established, the system enters the person tracking phase. Segmentation is performed based on the states; the centre of gravity (COG) is then passed to a Kalman Filter that predicts the position in the next frame. A new bounding box is formed around the predicted COG and passed to the HMM block to check the presence of a person again. If the person is still present, the parameters of the bounding box are again passed to the Kalman filter to make the next prediction and so on. If the person is no longer present, the system switches back to the person-detection phase. This is summarised in the system overview shown in Figure 4.

Note that while in the person-detection phase, the camera must be stationary for successful segmentation. However, once the system enters the tracking phase, panning and zooming are allowed. This is one of the strengths of this system as compared to many other tracking algorithms.

3.2. Implementation Considerations

For the purposes of this paper, we are only concerned with the extraction of the state-sequence from the observation values. As such, we have not considered the performance of this specific structure for the HMM as compared to others, nor trained the HMM ourselves. We have used model parameters supplied from some precursory work on the same system. [11]

Since we are concentrating on this processing unit, we have also pre-computed the transition probability values in the log-domain, and have assumed that the observation probabilities are in the log domain too. There is already much work in function approximation on FPGAs and so we can safely assume that a log-unit will be feasible in a full implementation. Despite the presence of hard-coded multipliers on the target device, we chose to stick with the log domain since it allows us to ignore problems with dynamic range which might force us to use floating-point.

The design was developed and implemented using the Handel-C compiler. This enabled us to test different levels of parallelism in a short amount of time with minimal extra effort. The targeted device was a Xilinx Virtex-II 6000 FPGA, on a Celoxica RC300 board, the target of our system implementation.

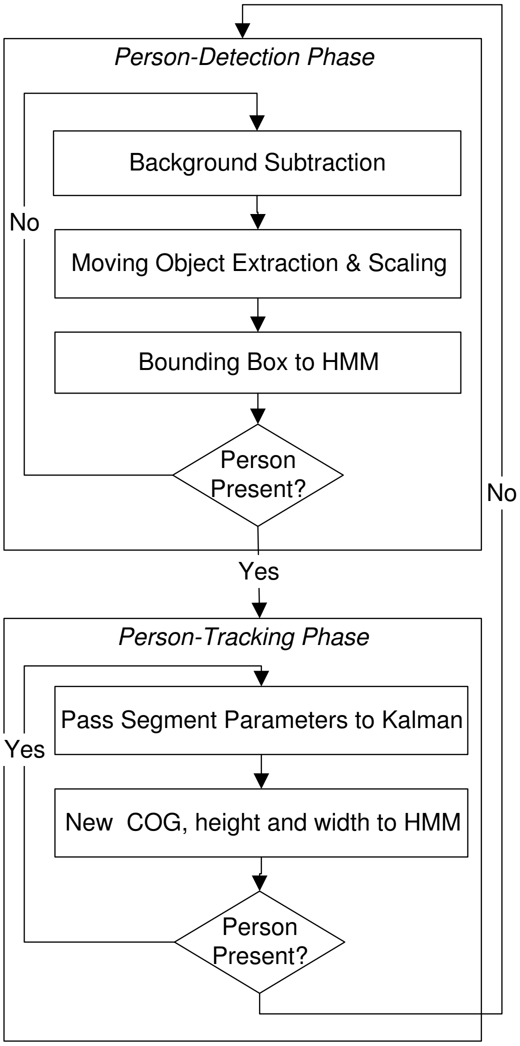

Figure 4. The Person-Tracking System

3.3. Dataflow considerations

In an implementation as complex as the Viterbi algorithm, organisation of data is paramount to an efficient design. Despite our design being much simpler than a general Viterbi decoder, there were a number of challenges in organising the delivery of data around the system.

The first important data is results from the previous timestep, $\delta_{t-1}(j)$. This is simply an array of 24 values that is copied from the current results, once each time the current timestep completes. The next data item is the observation probability. This is again an array of 24 values that changes each timestep. The required value is simply referenced by the number of the state currently being computed. The final and more complex type of data is the transition probabilities that are constant throughout. At first these were stored as a 24*x*24 array, and referenced by the values of *N* for this timestep and the previous, but this was too complex. Instead a much simpler approach was developed where each processing node has access to an array of tuples that contains the transition probabilities for the two predecessors, ignoring position, with predecessor selection was at a higher level. Hence the node itself is only the shaded region of Figure 3.

In implementing the parallelised versions, some other savings could be made. Consider first, that each hardware node only needs access to the transition elements for the states that it will calculate. More importantly, if one of the parallel nodes will not be computing any of states 1,7,13 or 19, then there is a saving since there is no need to select between two inputs as in the case of those nodes. This explains why the area requirement does not increase in proportion to the number of nodes, as all nodes above 4 are simpler in their circuitry.

Furthermore, as the level of parallelism increases, the control circuitry becomes more simple, so much so, that in the fully-parallel implementation, there is almost no control circuitry whatsoever. This is clear in the results.

3.4. Single-node Implementation

For this implementation a single calculation was implemented. In each timestep, control circuitry uses the node to calculate the results for each state, choosing the correct predecessors. The result are then shifted into a shift-register. Once all results had been computed for one timestep, the results are copied, in parallel, to the register holding previous results, ready for calculation of results in the next timestep.

This design takes 24 clock cycles to complete the state calculations for each timestep. The fastest clock rate achievable with the circuit is 36MHz.

3.5. Multi-node Implementations

Implementations were completed for 4, 8, 12 and 24 nodes in parallel. These designs take 6, 3, 2 and 1 clock cycle(s), to complete the calculations for one timestep, respectively.

In each case, the appropriate number of nodes is instantiated in parallel. Surrounding logic decides which data to pass to which node. Each node only needs access to whichever data it will process in the case of the transition probabilities only the necessary tuples were attached to each node.

Nodes	Slices	Cycle-time	Cycles/Res	M Res/s
1	972	27.033ns	24	1.5
4	2083	34.467ns	6	4.8
8	2271	30.570ns	3	10.9
12	1593	22.112ns	2	22.6
24	1425	14.953ns	1	66.9

Table 1. Implementation Results

The implementation for 24 nodes is much simpler than the others. The reason is that in the case of the 24 parallel nodes, each is hard-wired to the appropriate predecessor registers and transition values, and so there is no control circuitry as such. Since five of the transition probabilities are zero in our case, this removes one of the adders from those nodes. The 12-node version only has binary selections since it only runs for two clock-cycles. Hence there is a massive saving on the multiplexing of signals that causes it to be more area efficient that the 8-node implementation.

The implementations brought to our attention an interesting fact: that in the case of our design, the control circuitry was a significant part of the area. This is because a 1-bit adder uses the same amount of resources as a 2-way 1-bit select. Since in our designs the predecessor data is multiplexed into each node, this becomes significant. This is why there is a significant drop in area usage in the graphs from 8 to 12 nodes.

3.6. Implementation Results

Implementation results are summarised in Table 1 and Figures 5,6 and 7. From the graphs, it is clear that the 24-node implementation is most desirable. It is both faster than all other designs and smaller than all except the single-node implementation. However of importance too is the number of cycles needed for a complete result. This swings the result even more in favour of the 24-node implementation as seen in the throughput figures in the table. For reference a full Viterbi decoder in MATLAB, running on a Pentium 4, 2.4GHz machine, with the same data managed only 1000 results per second. We estimate that we require a rate of greater than 200,000 for a realtime implementation with 30fps video for the given state representation.

4. Conclusions

We have shown how, through taking into account the structure of the state representation for our HMM, it is possible to significantly simplify the computation of the state

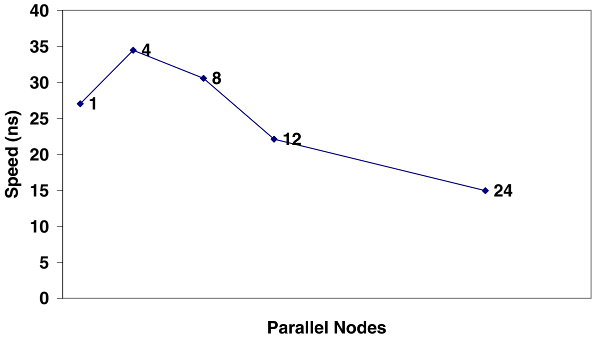

Figure 5. Speed vs. Number of Nodes

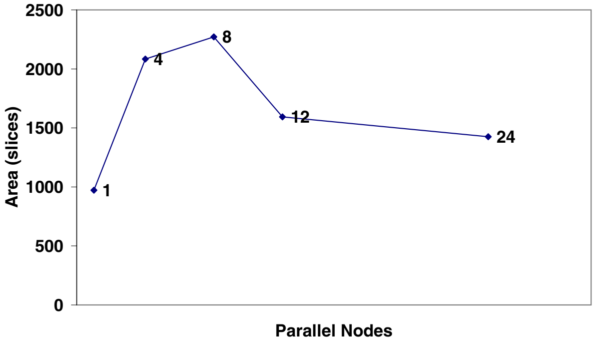

Figure 6. Area vs. Number of Nodes

sequence. The number of computations is reduced from $(N^2+1)\cdot T$ to $(2N+1)\cdot T$. We also explored the different levels of parallelism, and found that increasing the number of nodes not only drastically increases performance, but also has a positive impact on area usage. This is due to the control circuitry becoming simpler as more nodes are implemented in parallel.

4.1. Further Work

We will be investigating the HMM structure itself and deciding whether indeed this is the best state representation to have. Considering the increase in evaluation rate, we are less constrained in our selection now.

We are also investigating the logarithm block to precede the state decoding. We will investigate what effect inclusion of the logarithm will have on the speed of the different designs presented. It is envisaged that the more parallel designs will require a lot of computation for the logarithms, since they must be supplied in parallel too. This could mean selection of a design becomes less trivial.

References

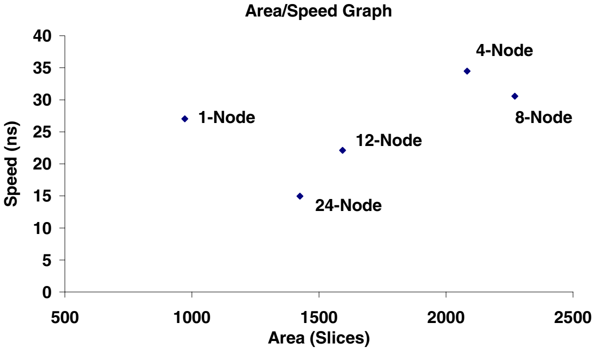

Figure 7. Area/Speed Graph for the Designs

[1] J.-G. Baek, S.-H. Yoon, and J.-W. Chong. Memory efficient pipelined viterbi decoder with look-ahead trace back. In *The 8th IEEE International Conference on Electronics, Circuits and Systems, 2001. ICECS 2001.*, volume 2, pages 769–772 vol.2, 2001.

[2] M. Bicego and V. Murino. Investigating hidden markov models' capabilities in 2d shape classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(2):281–286, 2004.

[3] H. Breit and G. Rigoll. Improved person tracking using a combined pseudo-2d-hmm and kalman filter approach with automatic background state adaptation. In *International Conference on Image Processing, 2001. Proceedings.*, volume 2, pages 53–56 vol.2, 2001.

[4] H. Breit and G. Rigoll. A flexible multimodal object tracking system. In *International Conference on Image Processing, 2003. Proceedings.*, volume 3, pages III–133–6 vol.2, 2003.

[5] S.-S. Kuo and O. Agazzi. Keyword spotting in poorly printed documents using pseudo 2-d hidden markov models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(8):842–848, 1994.

[6] S. Melnikoff. *Speech Recognition in Programmable Logic*. Ph.d., The University of Birmingham, 2003.

[7] A. Nefian. *A Hidden Markov Model-Based Approach for Face Detection and Recognition*. Ph.d., Georgia Institute of Technology, 1999.

[8] L. Rabiner and B. Juang. An introduction to hidden markov models. *IEEE ASSP Magazine*, 3(1):4–16, 1986.

[9] G. Rigoll, S. Eickeler, and S. Muller. Person tracking in real-world scenarios using statistical methods. In *Fourth IEEE International Conference on Automatic Face and Gesture Recognition, 2000. Proceedings.*, pages 342–347, 2000.

[10] G. Rigoll, B. Winterstein, and S. Muller. Robust person tracking in real scenarios with non-stationary background using a statistical computer vision approach. In *Second IEEE Workshop on Visual Surveillance, 1999.*, pages 41–47, 1999.

[11] M. Yaqoob. *Object Tracking Algorithms and Implementations*. M.sc., Imperial College London, 2003.

[12] Y. Zhu and M. Benaissa. A novel acs scheme for area-efficient viterbi decoders. In *Proceedings of the 2003 International Symposium on Circuits and Systems. ISCAS '03.*, volume 2, pages II–264–II–267 vol.2, 2003.