A Low Memory Bandwidth Gaussian Mixture Model (GMM) Processor for 20,000-Word Real-Time Speech Recognition FPGA System

Kazuo Miura, Hiroki Noguchi, Hiroshi Kawaguchi, and Masahiko Yoshimoto
Kobe University, Kobe, 657-8501 Japan
k-miura@cs28.cs.kobe-u.ac.jp

Abstract

We propose a GMM processor for large vocabulary real-time continuous speech recognition. This processor achieves low operating frequency and low memory bandwidth using parallelization and vector look-ahead schemes, which are suitable to FPGA implementation. We designed the proposed processor on a Celoxica RC250 FPGA board, and confirmed that the required frequency and memory bandwidth for real-time operation are reduced by 89.8% and 84.2%, respectively. The 20,000-word real-time GMM computation is made at a frequency of 30.4 MHz and memory bandwidth of 47 Mbps, on the prototype.

1. Introduction

There are some software-based recognition systems, but they are not suitable for mobile devices because those solutions require high-performance processors that consume far more power than mobile processors [1, 2]. Therefore, a hardware approach, such as a VLSI or FPGA, which is superior to the general software-based implementation in terms of power and speed, is needed.

Yoshizawa et al. implemented a real-time recognition system onto a VLSI for 800-word isolated word recognition [3]. Edward C. Lin et al. investigated FPGA implementation for 1,000-word continuous speech; but it did not run in real time [4]. S J Melnikoff et al. implemented real-time continuous recognition system, but the recognition accuracy was impractical less than 60% [5]. As far as the authors know, large vocabulary (more than 5,000 words) real-time continuous speech recognition (LVRCSR) in high accuracy using hardware approaches is not realized yet.

In LVRCSR, computation of GMM occupies more than 60% of whole execution time. In this paper, in order to achieve LVRCSR in practical accuracy, we pr embed a novel GMM processor on an FPGA.

2. Speech recognition overview

2.1. Hidden Markov model (HMM)

The HMM algorithm is de facto of the speech recognition. The HMM is shown in Fig. 1. It is modeled as follows: π is initial-state probability, a_{ij} is transition probabilities from state i to state j and $b_j(x_t)$ is output probability density functions of GMM), where x_t is a feature vector extracted from speech. For instance, if we assume that a feature vector sequence is x_1, x_2, x_3 and transition sequence is q_1, q_2, q_3, q_4, then the probability of the transition from q_1 to q_4, $P(q_1 \rightarrow q_4)$, can be calculated with (1).

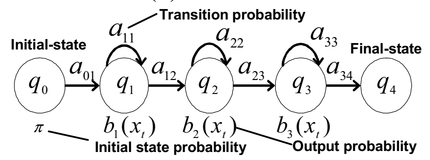

Figure 1. Left-right HMM.

$$P(q_1 \rightarrow q_4) = \pi \times a_{01} b_1(x_1) \times a_{12} b_2(x_2) \times a_{23} b_3(x_3) \times a_{34} \quad (1)$$

In the HMM algorithm, each HMM corresponds to a phone. Each word is expressed as a sequence of phones, and each sentence is represented as sequence of words.

2.2. Time-synchronous Viterbi search

Initialization:

$$\delta_0(0) = \log \pi \quad (2)$$

Recursion:

$$\delta_t(j) = \max_{i=j-1,\, j} [\delta_{t-1}(i) + \log a_{ij}] + \log b_j(x_t)$$
$$\text{for } 1 \le t \le T, 1 \le j \le N_{state} \quad (3)$$

Termination:

$$P(w \mid x_1, x_2, \cdots, x_T) = \max_{N_f} [\delta_T(i)], \text{ for } t = T \quad (4)$$

The above formulas show the log-Viterbi algorithm. To prevent from underflow, logarithms are taken. Here, T is the number of frames, N_{state} is the number of all

HMM states, N_f is the states set that correspond to word end, and i and j are state indexes. $\delta_t(j)$ is a likelihood value at a time index t and state j. w is a recognition output sentence. First in the HMM algorithm, the likelihood value is initialized as $\log[\pi]$.Speech is divided into frames (15-25 ms), and a feature vector is calculated in each frame. Equation (3) shows that, once a feature vector is obtained, each state in the HMM move to the next state that maximizes the likelihood value. This is the reason why the transition sequence is uniquely determined.

2.3. N-gram model

To achieve high-accuracy recognition, a language model is added to (3) as $P(w)$. w is a sentence and a language model represents grammatical accuracy of sentences. Generally N-gram model is used.

2.4. Speech recognition flow with HMM

Fig. 2 shows a speech recognition flow with HMM. The following items describe concrete steps.

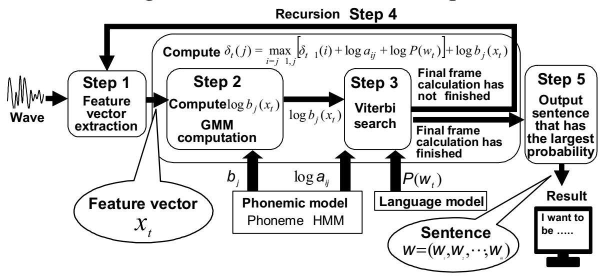
Recursion Step 4
Compute $\delta_t(j) = \max_{i=j-1,j}[\delta_{t-1}(i) + \log a_{ij} + \log P(w_t)] + \log b_j(x_t)$
Wave
Step 1
Feature vector extraction
Step 2
Compute $\log b_j(x_t)$
GMM computation
$\log b_j(x_t)$
Step 3
Viterbi search
Final frame calculation has not finished
Final frame calculation has finished
Step 5
Output sentence that has the largest probability
b_j
$\log a_{ij}$
$P(w_t)$
Feature vector x_t
Phonemic model
Phoneme HMM
Language model
Result
Sentence $w=(w_1,w_2,\cdots,w_m)$
I want to be .....

Figure 2. Speech recognition flow

1. *Feature vector extraction*: a feature vector is extracted on a frame-by-frame basis.
2. *GMM calculation*: compute GMM probabilities $\log[b_j(x_t)]$ of all active states.
3. *Viterbi beam search*: calculates $\delta_t(j)$ by using the GMM probabilities and N-gram model, $P(w_i)$.
4. *Recursion*: repeats Steps 1-3 in all frames.
5. *Output*: after the final frame, the transition sequence of the word-end state that has the maximum likelihood value is output as the result of recognition.

2.5. Computation Time Analysis

To identify the highest-load part in speech recognition, we estimated each execution time by using a well-known Japanese speech recognition system software, Julius [1]. Julius has two phases: The first phase (hereafter, we call "the 1-path") is based on a frame synchronized with a Viterbi beam search. The second phase (hereafter, we call "the 2-path") is based on a heuristic search, which reuses the output of the 1-path as a heuristic function.

Table 1. Parameters and models used in Julius.

	1-path	2-path
Phonemic model	Gaussian 16 mixture tri-phone model	
Language model	20000 word bi-gram	20000 word tri-gram

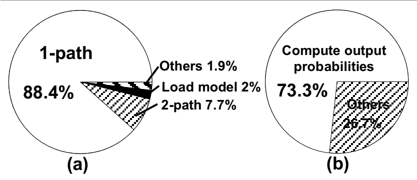

Figure 3. Computation time breakdowns in (a) whole flow and (b) 1-path.

As well, Table 1 shows the parameters and models in the estimation. Fig. 3 (a) illustrates that the 1-path dominates 88.4% of the total computation time, in which computing output probabilities occupies 73.3%. For LVRCSR recognition, minimizing the computation time of the output probabilities is effective.

3. Design of GMM processor

3.1. GMM computation

The GMM computation obtains $\log[b_j(x_t)]$ from a feature vector x_t and parameters of a GMM, which is used in the Viterbi search algorithm. As expressed in (5), $\log[b_j(x_t)]$ is expressed as logarithm of a sum of the Gaussian distribution multiplied by weight functions. We assume Σ_i is diagonal matrix and simplify it.

$$\log b_j(x_t) = \sum_i^{mix} \lambda_i N(x_t, \mu_i, \Sigma_i) \tag{5}$$

$$= \log\left[\sum_i^{mix} \lambda_i \left[\frac{1}{(2\pi)^{\frac{P}{2}} |\Sigma_i|^{\frac{1}{2}}} \exp\left\{-\frac{1}{2}(x_t - \mu_i)' \Sigma_i (x_t - \mu_i)\right\}\right]\right]$$

$$= add\log\left[w_{ij} + \sum_{s=1}^{P} (x_{ts} - \mu_{ijs})^2 \sigma_{ijs}\right] \tag{6}$$

$$w_j = \log \lambda_i + \log\left[\frac{1}{(2\pi)^{\frac{P}{2}} \left(\prod_{s=1}^{P} \Sigma_{ijs}\right)^{\frac{1}{2}}}\right] \quad add\log[X_i] = \log \sum_{i=1}^{mix} X_i$$

where each parameter is as follows: $b_j(x_t)$ is a GMM PDF, N is a Gaussian distribution PDF, P is the number of dimensions in a feature vector, mix is the number of mixtures in the GMM, x_t is a feature vector, μ is a mean parameter, Σ is a variance-covariance matrix, and λ is a weight function. w_{ij} is a constant number, and can be computed before speech recognition, offline. Equation (6) indicates that the GMM computation at one dimension consists of one addition, one subtraction, two multiplications, *P* summations, and taking a logarithm of them.

3.2. Implementation of GMM computation

The GMM calculations occupy over 60% of whole execution time as shown in Section II. To achieve speech recognition in real time yet at a lower operating frequency than an FPGA, we propose parallel architecture with low memory bandwidth. Our proposed scheme features the following three points.

- Parallel computing of Gaussian distributions as to the number of GMM mixtures.
- Parallelization in taking logarithms based on a look-up table
- Pipeline architecture for reading Gaussian distribution parameters and calculating them.

In the first feature, the parallelism can be theoretically increased up to the number of the mixtures in the GMM; however, it linearly increases memory bandwidth. For this reason, in our FPGA implementation, the parallelism in computing the Gaussian distributions is limited to four.

As the second one, we prepare 2-input addlog units. The 2-input addlog unit calculates an approximate logarithmic value of a sum of two inputs. For instance, to carry out four "addlog"s, four data are divided into two groups; each group is input to two 2-input addlog units, and individually calculated at the same time. The two 2-input addlog units output two results. Repeating this operation, we can obtain a desired output for any number of data. This way reduces computation cycles. The number of parallelism in taking logarithms is four by the restriction of our FPGA.

The third one means that, in our dedicated hardware, reading memory and calculating Gaussian distributions are simultaneously performed.

3.3. Vector look-ahead scheme

The GMM calculations require high memory bandwidth because many GMM parameters should be read from memory. For a low memory bandwidth suitable to an FPGA, we propose a novel vector look-ahead scheme using locality of GMM data.

Each state in the HMM has a specific GMM. For its GMM calculation, it is necessary to load GMM data from memory. However, each phonemic HMM has self transition, and fortunately the GMM data used in the present frame will be reused in the next frame, at high probability. This probability reaches more than 90%.

In the vector look-ahead scheme, several feature vectors are buffered in advance, and their output probabilities are computed in parallel. Then, the answers are stored in cache. If a duplicated state appears at the next frame, the answer stored in the cache is outputted. This scheme reduces the memory bandwidth and cycles. The maximal number of vector look-ahead depth is seven and their feature vectors are 24-bit fixed point accuracy. This is limited by the maximum circuit scale allowed on the FPGA.

3.4. Architecture

We utilized an RC250 FPGA board produced by Celoxica. Fig. 4 shows the proposed architecture implemented on RC250. Input data are the feature vectors and the state ID that decides which GMM is used. Output is the output probability that corresponds to the state ID.

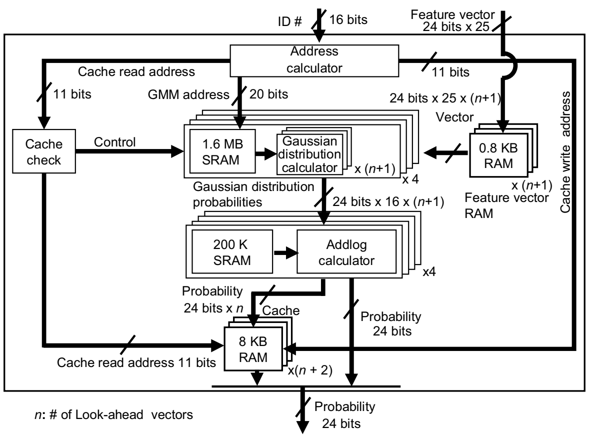

Figure 4. Proposed architecture.

First a first vector and *n* look-ahead vectors are stored in the feature vector RAM. After this, the oldest vector is overwritten with a new vector. Next the address calculator calculates a cache read address, the GMM address and the cache write addresses that point where the calculated output probabilities will be cached. Then the cached data are checked. If hit, the cached probability is output as a result. If not, the output probabilities of contiguous *n*+1 vectors are computed in parallel. Finally the computation result that corresponds to the present frame is output. The other probabilities regarding *n* look-ahead vectors are stored in the cache.

4. Implementation results

To verify the implementation and compare with a PC, we constructed all operations concerning the speech recognition. However, the FPGA implementation only includes the computation of the GMM. So, the other operations in the speech recognition are implemented as software using Julius. The models and parameters have been listed in Table 1.

4.1. Accuracy degradation by fixed point

We adopted the fixed point for simple hardware; however, it might give a negative impact on the recognition accuracy. The recognition accuracy degradation is shown in Table. 2. In 24 bit fixed-point, the degradation is up to 0.3%.

Table 2. Accuracy affected by fixed point.

The # of bits	Accuracy
PC (Floating point)	92.9
24	92.6

4.2. Gate utilization

Implementing the parallel computation and vector look-ahead scheme augments a hardware size. Table. 3 shows the number of NAND gates used, when the parallel computation is implemented and the vector look-ahead depth is changed.

Table 3. Gate utilizations.

Parallel computation	Don`t	Do	Do	Do	Do	Do
The # of look-ahead	0	0	1	3	5	7
The # of NAND gates [10^6 gates]	0.55	0.83	1.5	2.4	3.4	4.3

4.3. Frequency and memory bandwidth

Fig. 5 shows the required frequency for real-time operation. As well as Table. 3, Fig. 5 shows the both cases that the parallelization is implemented and not. The vector look-ahead depth is also changed.

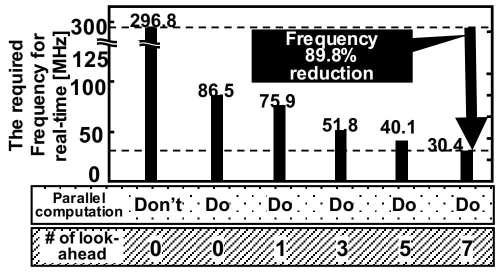

Figure 5. Required frequencies.

From the figure, the required frequency is dramatically reduced to 86.5 MHz (70% reduction). This is because the parallelism is four. In addition, as the vector look-ahead depth is increases, the frequency is decreased. When the vector look-ahead depth is seven, the required frequency is suppressed to 89.8%.

For the vector look-ahead scheme, we can also reduce the memory bandwidth as shown in Fig. 6, because the cache memory reduces the times of GMM data loading. The scheme reduces 84.2% when the depth is seven.

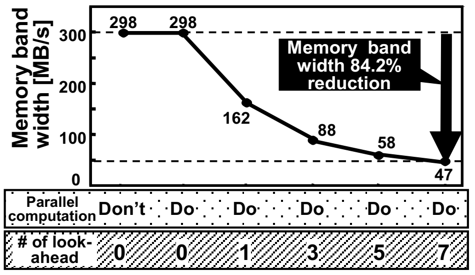

Figure 6. Memory bandwidths.

5. Summary

We proposed a novel FPGA implementation of the GMM computation, with a parallelization and vector look-ahead schemes. The required frequency and memory bandwidth for real-time operation are reduced by 89.8% and 84.2%, respectively, compared with the case without these schemes. As a result, our architecture achieves real-time GMM computing for 20,000 words speech recognition FPGA system. The operating frequency is 30.4 MHz and memory bandwidth is 47 Mbps.

References

[1] A. Lee, T. Kawahara and K. Shikano, "Julius – an open source real-time large vocabulary recognition engine", *Proc. European Conference on Speech Communication and Technology*, Denmark, 2001, pp. 1691-1694.

[2] The CMU Sphinx Speech Recognition Engines http://cmusphinx.sourceforge.net/html/cmusphinx.php

[3] S. Yoshizawa, N. Wada, N. Hayasaka and Y. Miyanaga, "Scalable Architecture for Word HMM-Based Speech Recognition and VLSI Implementation in Complete System", *IEEE Trans. on Circuits and Systems*, USA, 2006, pp. 70-77

[4] E. C. Lin, K. Yu, R. A. Rutenbar, and T. Chen, "A 1000-Word Vocabulary, Speaker-Independent, Continuous Live-Mode Speech Recognizer Implemented in a Single FPGA", *International Symposium on Field-Programmable Gate Arrays (FPGA)*, USA, 2007, pp. 60-68.

[5] S. J. Melnikoff, S. F. Quigley and M. J. Russell, "Performing Speech Recognition on Multiple Parallel Files Using Continuous Hidden Markov Models on an FPGA", *Proc. IEEE international conference on Field Programmable Technology (FPT), Hong Kong*, 2002, pp. 399-402.