HARDWARE-SOFTWARE CO-DESIGN OF AUTOMATIC SPEECH RECOGNITION SYSTEM FOR EMBEDDED REAL-TIME APPLICATIONS

*V.Rajya Lakshmi[1], P. Venkat Rao[2]
[1]PG Student (M.Tech VLSI&ES), Dept. of ECE, DRK College of Engineering & Tech, Hyderabad, AP, India
[2]Assistant Professor, Dept. of ECE, DRK College of Engineering & Tech, Hyderabad, AP, India

Abstract: In this paper we propose "Hardware-software Co-design of automatic speech recognition system for embedded real-time applications". ASR has been widely used in human–machine interaction, such as mobile robots, consumer electronics, and manipulators in industrial assembly lines, automobile navigation systems, and security systems. Here we use ARM 32-bit micro controller which supports features and algorithms for designing of automatic speech recognition system for real time applications. To recognize the speech we use HMM algorithm. Its main purpose is to convert a speech signal into a sequence of acoustic feature vectors. The entire speech signal is segmented into a sequence of shorter speech signals known as frames.

Keywords: ARM, HMM, Automatic speech recognition (ASR), embedded system, hardware–software codesign.

1. INTRODUCTION

Automatic speech recognition (ASR) on embedded platforms has been gaining its popularity. The existing technology of automatic speech recognition is done by using speech recognition modules which recognizes words that are predefined internally and the words should be pronounced clearly and then module recognizes word corresponding task is done. But the main disadvantage of these systems are hardware complexity i.e. we require voice modules to do these tasks and to recognize a word these module should kept very close to mouth and also we should carry these module everywhere task should be done. We can overcome this by usuing ARM32-bit micro controller which supports features and algorithms for designing of automatic speech recognition system for real time applications.

automatic speech recognition (ASR) on embedded platforms has been gaining its popularity. ASR has been widely used in human–machine interaction, such as mobile robots, consumer electronics, and manipulators in industrial assembly lines, automobile navigation systems, and security systems. More sophisticated ASR applications with larger vocabulary sizes and more complex knowledge sources are expected in the future. As a result, the demand for high performance, accurate, and fast embedded ASR is increasing.

In many embedded ASR systems, a pure software-based approach is taken by developers . This approach enables fast deployment of ASR-based applications. However, the timing performance is constrained by the processing power and memory bandwidth of the target platforms. As a result, word accuracy is often compromised for better timing performance.

2. HARDWARE–SOFTWARE COPROCESSING SYSTEM

The ASR algorithm is partitioned into three main parts: feature extraction, GMM emission probability calculation, and Viterbi search. The speech recognizer is first implemented in software where the 16-b fixed-point implementation of the recognizer is compared with the floating-point implementation. The experimental results show that there is no degradation in recognition accuracy in the fixed-point implementation. Hence, the fixed-point system is chosen as our baseline system for time profiling. It shows that about 69% of the total elapsed time is spent on GMM computation. The proportions of time spent on feature extraction and Viterbi search are 7% and 24%, respectively. Since GMM computation is the most computationally intensive part, a hardware accelerator is designed in order to speed up this part of the ASR algorithm.

3. SYSTEM ARCHITECTURE

The architecture of the hardware–software coprocessing system is shown in Fig. 1. The system consists of an Altera Nios II processor core and a GMM hardware accelerator. The Nios II processor acts as the control unit of the entire system.

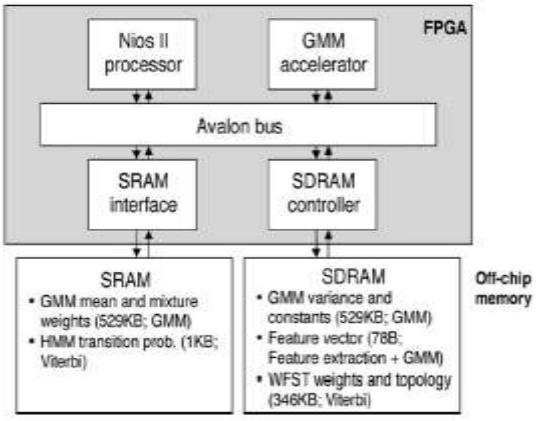

Figure 1 System architecture of the hardware–software co processing recognizer with the GMM hardware

Feature extraction and Viterbi search are implemented in software. When the system needs to perform a GMM

International Journal of Instrumentation Electrical Electronics Engineering
p- ISSN: 2321 – 0621 e- ISSN: 2321- 063X

calculation, the processor instructs the accelerator to carry out the computation.

The accelerator returns the computation result to the Nios II core. The entire coprocessing system is synthesized on an Altera Stratix II EP2S60F672C5ES field-programmable gate array (FPGA).

4. GMM Emission Probability Hardware Accelerator

Datapath: The GMM hardware accelerator calculates the log emission probability of an observation vector given an HMM state. Given an observation feature vector ot, the emission probability function in an HMM state j is modeled by a sum of weighted Gaussian mixtures.

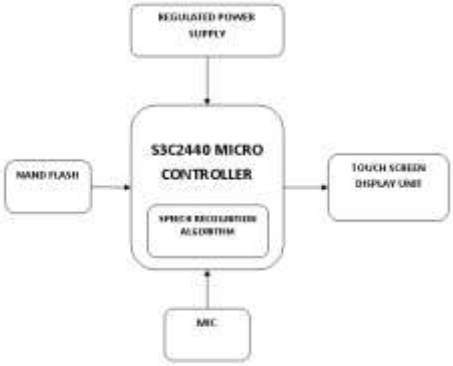

Figure 2 Block Diagram

Timing Profile: After synthesis and place and route, the proposed system is implemented on the target FPGA board. The first experiment is to investigate the relationship between the speedup in GMM calculation and the number of parallel computation units (N). The aim is to find the smallest number of computation units with maximum speedup. Fig. 2 shows the number of clock cycles for GMM calculation versus the number of computation units. The task is the Resource Management (RM1) task, which consists of 1200 test utterances. The vocabulary size is 993. Triphone HMM models with three emitting states and four Gaussian mixtures per state are trained on 2880 utterances. Acoustic features are 39-D MFCCs with the zeroth coefficient plus their delta and delta–delta coefficients.

The language model is word-pair grammar (bigram). The experimental result shows that the speedup reaches the maximum when N ≥ 10. Compared with only one computa tion unit, the speedup using ten units is about 3.73 times. As a result, the proposed architecture includes ten computation units. The reason for this is that the log-add unit requires four clock cycles to complete each log-add operation.

The GMM hardware accelerator significantly shortens the time for GMM computation. The speed-up is about 108 times. The total decoding time reduces from 4.70 to 1.49 s. The real-time factor improves from 1.87 to 0.59 which is well below 1.00.

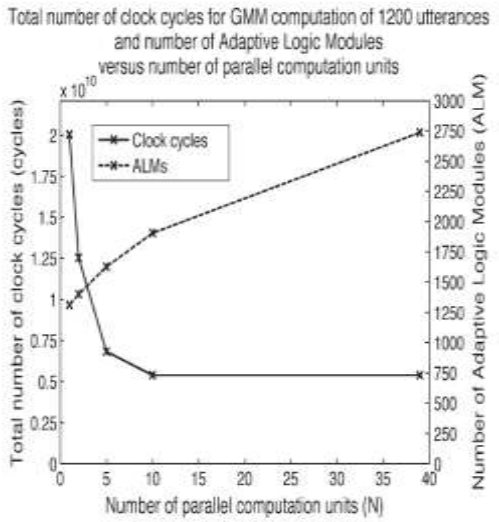

Figure 3 Total number of clock cycles for GMM computation of 1200 utterances and the number of ALMs used by the GMM accelerator versus the number of parallel computation units (N = 1, 2, 5, 10, 39). ALMs are the logic elements of a Stratix II FPGA.

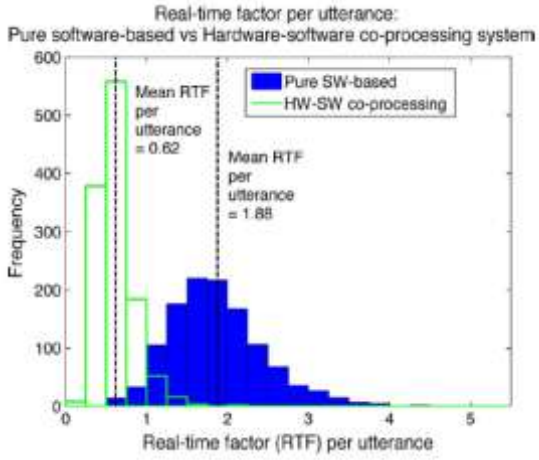

Figure 4 Real-time factor of 1200 utterances in two different systems: Pure software-based system versus Hardware–software coprocessing system. Pruning beamwidth = 170.

The proposed system is tested on the entire test set of the RM1 corpus consisting of 1200 utterances and the results are shown in Fig.3.

The average real-time factor improves from 1.88 to 0.62. The speed-up is about 3.03 times. The average real-time factor of the hardware–software coprocessing system is below one, which suggests that the decoding time is shorter than the speech duration. There is a significant shift of the real-time factors from above one to below one. The coprocessing system significantly improves the timing performance. n terms of word accuracy, the GMM accelerator is the exact implementation of the algorithm. Hence, the word accuracy rate is 93.33% which is the same as that of the pure software-based system.

ADAPTIVE PRUNING

One method for lowering the number of active tokens is to adopt a tighter pruning beamwidth. However, it will introduce search errors which often decrease the recognition accuracy. Our goal is to reduce the decoding time of those utterances which

have a relatively greater real-time factor, while keeping the recognition accuracy of the other utterances. In order to fulfil this goal, an adaptive pruning scheme is proposed, where the pruning beamwidth is adaptive according to the number of active tokens.

5. ALGORITHM

Fig. 5 shows the pseudocode of the ASR algorithm with adaptive pruning. In the beginning, the beamwidth is initialized to a value. Before token passing, the algorithm modifies the pruning beamwidth according to the number of active tokens, n(˜Qt). If the number of tokens is greater than a threshold, τupper, a tighter beamwidth is adopted. The beamwidth is decreased by a certain amount denoted by δ .

The proposed pruning scheme is more flexible than the narrow and fixed pruning scheme. The number of active tokens is often time varying in the duration of an utterance. The fixed pruning scheme applies a tight beamwidth throughout the entire utterance regardless of the number of active tokens. On the other hand, the adaptive scheme allows relaxation of the beamwidth in parts of the utterance where the workload is less heavy.

Algorithm 3 Speech recognition algorithm with adaptive beam pruning

```
/* Q̃_t is a set of HMM states which have tokens at time t */
Q̃_1 ← Q_word−start
score_{q,1} ← 0 for all q ∈ Q̃_1
pruning_beamwidth ← original_beamwidth

for t = 1 to T do
    o_t ← Feature_extraction(Frame_t)

    max_score ← max(score_{q,t}) for all q ∈ Q̃_t

    if n(Q̃_t) > τ_upper then
        pruning_beamwidth ← pruning_beamwidth − δ
    else if n(Q̃_t) < τ_lower then
        if pruning_beamwidth < original_beamwidth then
            pruning_beamwidth ← pruning_beamwidth + δ
        end if
    end if

    pruning_threshold ← max_score − pruning_beamwidth
    Q̃_{t+1} ← {}

    for all q ∈ Q̃_t do
        if score_{q,t} > pruning_threshold then
            log_emis_prob ← Emission_prob_calc(o_t, q)
            V ← Viterbi_search(log_emis_prob, q, t)
            Q̃_{t+1} ← Q̃_{t+1} ∪ V
        end if
    end for
end for

Q ← Q̃_{T+1} ∩ Q_word−end
best_token ← argmax_{q∈Q}(score_{q,T+1})
```

Figure 5 Speech recognition algorithm with adaptive beam pruning.

6. SIMULATION RESULTS

In terms of implementation, the proposed adaptive scheme is simpler than histogram pruning. Implementing histogram pruning requires a sorted list of the token scores. For each token, the recognizer needs to perform an insertion sort which involves searching for the token's ranking in a sorted list of the previously iterated token scores. Maintaining the tokens in a sorted order is computationally intensive.

Timing Profile

Fig. 6 shows the real-time factor of the coprocessing system. Fixed beam pruning and adaptive beam pruning are compared. The beamwidth is held constant at 170 for the fixed beam-pruning scheme. In adaptive beam pruning, the original_beamwidth variable is also set to 170. The thresholds, τlower and τupper, are 1900 and 2300, respectively. The beamwidth adjustment value is 10 (δ = 10). These parameters are determined empirically.

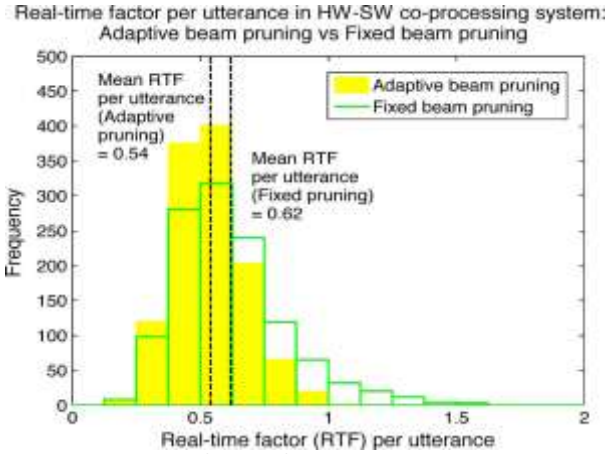

Figure 6 Real-time factor of 1200 utterances in hardware–software coprocessing system. Adaptive beam pruning versus Fixed beam pruning.

In the fixed beam-pruning scheme, about 94% of the utterances have a real-time factor below one. When the adaptive beam-pruning scheme is used, this percentage increases to 99.75%. Only 3 out of 1200 utterances have a real-time factor above one. Compared with the fixed beam-pruning scheme, there is a small degradation in recognition accuracy which decreases from 93.33% to 93.16%.We have also tried to tighten the adaptive pruning scheme by adjusting τupper and τlower to smaller values (τupper = 1700, τlower = 1250), so that the real-time factors of all the utterances are below 1. The word accuracy rate reduces to 92.62%.

PERFORMANCE COMPARISON

Table I compares the performance of our proposed system with other existing systems. The clock frequency of the proposed system is determined by the maximum working frequency (fmax) of the GMM accelerator which is 120 MHz.

The pure software-based systems require a higher number of clock cycles for performing the same task[3]. As a result, a higher clock frequency is needed. On the other hand, pure hardware based systems and hardware–software coprocessing systems can run the same task at lower clock frequency[5].

As shown in the table, the word accuracy rate of our proposed system is within the range of the other systems. The proposed system performs better than PocketSphinx and In Silico Vox systems. The AT&T system from shows slightly better word accuracy but

the acoustic features are stored in files and accessed by the StrongARM platform from a PC.

Table 1 Performance of recently developed embedded speech recognition systems and our proposed system on the 993-word rm1 task

System	CPU/Platform	Clock frequency (MHz)	Word accuracy rate (%)	Real-time factor
Pure software-based system				
PocketSphinx [8]	StrongARM	206	88.05	0.87
AT&T [9]	StrongARM	206	≈ 94	1.00
Pure hardware-based system				
In Silico Vox [10], [11]	Xilinx Virtex-II Pro XC2VP30 FPGA	50	89.10	2.50
Speech Silicon [12]	Xilinx Virtex-4 XC4VSX35 FPGA	100	N/A	N/A
Hardware-software co-processing system				
Seoul National University [13], [14]	MicroBlaze on Xilinx Virtex-4 XC4VSX35 FPGA	100	96.29*	N/A
Our system with fixed beam pruning	Nios II on Altera Stratix II EP2S60F672C3ES FPGA	120	93.53	0.62
Our system with adaptive beam pruning	Nios II on Altera Stratix II EP2S60F672C3ES FPGA	120	93.16	0.54

*The test set contains only 300 utterances.

It suggests that the acoustic features may not be generated by the StrongARM platform. Therefore, in order to achieve the same real-time factor including feature extraction, a tighter beamwidth may be needed, which may decrease the word accuracy. For the Seoul National University system, the word accuracy rate is higher than the other systems. However, their results are based on only 300 utterances, whereas there are 1200 test utterances in our experiments.

The real-time factors of the proposed system, as shown in Table I, are calculated by dividing the total decoding time by the speech duration of the entire test corpus. Table III shows that the real-time factors of our proposed system are well below 1.00, and they are much better than those of the other reported systems.

For many ASR applications, it is not necessary to ensure that the real-time factors of all the utterances are below 1.00 as the user can tolerate a small time delay in machine response. However, for more complex applications where the system needs to perform multiple tasks with ASR, it will be beneficial if ASR can be done as quickly as possible, so that the remaining time can be used for other tasks. As the real-time factors of our proposed system are well below 1.00, it indicates that the proposed system is more capable of multitasking with ASR than other reported systems.

The Seoul National University system is also a hardware–software coprocessing system. However, the realtime factor of their system is not shown and thus its timing performance cannot be compared with our proposed system

4. CONCLUSIONS

The paper “Hardware-software Co-design of automatic speech recognition system for embedded real-time applications” has been successfully designed and tested. It has been developed by integrating features of all the hardware components and software used. Presence of every module has been reasoned out and placed carefully thus contributing to the best working of the unit. Secondly, using highly advanced ARM9 board and with the help of growing technology the project has been successfully implemented.

The performance of the proposed system is sufficient for a wide range of speech-controlled applications. For more complex applications which involve multiple tasks working with ASR, further improvement of timing performance, for example, by accelerating the Viterbi search algorithm, might be required. The proposed coprocessing architecture can easily accommodate additional hardware accelerators.

Acknowledgements

The authors would like to thank the anonymous reviewers for their comments which were very helpful in improving the quality and presentation of this paper.

References

[1] A. Green and K. Eklundh, “Designing for learnability in human–robot communication,” IEEE Trans. Ind. Electron., vol. 50, no. 4, pp. 644–650, Aug. 2003.
[2] B. Jensen, N. Tomatis, L. Mayor, A. Drygajlo, and R. Siegwart, “Robots meet humans—Interaction in public spaces,” IEEE Trans. Ind. Electron., vol. 52, no. 6, pp. 1530–1546, Dec. 2005.
[3] N. Suresh K et al., (2011), “Effect of Interrupt Logic on Delay Balancing Circuit”. International Journal of Computer Applications 27(4):26-30, August.
[4] A. Chatterjee, K. Pulasinghe, K. Watanabe, and K. Izumi, “A particleswarm- optimized fuzzy-neural network for voice-controlled robot systems,” IEEE Trans. Ind. Electron., vol. 52, no. 6, pp. 1478–1489, Dec. 2005.
[5] N. Suresh Kumar et al., “A New Method to Enhance Performance of Digital Frequency Measurement and Minimize the Clock Skew”,vol 11. No 10. IEEE Sensor J, 2011.
[6] K. Saeed and M. Nammous, “A speech-and-speaker identification system: Feature extraction, description, and classification of speech-signal image,” IEEE Trans. Ind. Electron., vol. 54, no. 2, pp. 887–897, Apr. 2007.
[7] E. Bocchieri and D. Blewett, “A decoder for LVCSR based on fixed-point arithmetic,” in Proc. ICASSP, 2006, pp. 1113–1116.
[8] E. Lin, K. Yu, R. Rutenbar, and T. Chen, “Moving speech recognition from software to silicon: The In Silico Vox Project,” in Proc. Interspeech, 2006, pp. 2346–2349.
[9] H. Lim, K. You, and W. Sung, “Design and implementation of speech recognition on a softcore based FPGA,” in Proc. ICASSP, 2006, pp. 1044–1047.

Biographies

V.Rajya Lakshmi is a PG Student. She got her B.Tech degree in Electronics and Communication Engineering from ANURAG engineering college, JNTUH. She is doing her M.Tech from DRK College of Engineering & Tech, Hyderabad with specialization in VLSI&ES .

P.Venkata Rao is Assistant Professor, Dept. of ECE, DRK College of Engineering & Tech, Hyderabad.