International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 1 Issue 7, September - 2012

Voice Based Robo Using Automatic Speech Recognization System

M. Sravan Kumar
M.tech(PG student)
GITAS

B.Chinna rao
Prof.&Head,Dept.of ECE
GITAS

P.M.Francis
Asst.Prof. in Dept. of ECE
GITAS

Abstract:* We present a hardware–software coprocessing speech recognizer for real-time embedded applications. The system consists of a standard microprocessor and a hardware accelerator for Gaussian mixture model (GMM) emission probability calculation implemented on a field-programmable gate array. The GMM accelerator is optimized for timing performance by exploiting data parallelism. In order to avoid large memory requirement, the accelerator adopts a double buffering scheme for accessing the acoustic parameters with no assumption made on the access pattern of these parameters. Experiments on widely used benchmark data show that the real-time factor of the proposed system is 0.62, which is about three times faster than the pure software-based baseline system, while the word accuracy rate is preserved at 93.33%. As a part of the recognizer, a new adaptive beam-pruning algorithm is also proposed and implemented, which further reduces the average real-time factor to 0.54 with the word accuracy rate of 93.16%. The proposed speech recognizer is suitable for integration in various types of voice (speech)-controlled applications.

***Index Terms*—Automatic speech recognition (ASR), embedded system, hardware–software codesign, real-time system, softcore-based system.**

I. INTRODUCTION

RECENTLY, 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 [1]–[3], consumer electronics [4], manipulators in industrial assembly lines [5], automobile navigation systems [6], and security systems [7]. 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.

At another extreme, a speech recognizer can be tailor-made in a *pure hardware-based* system [10]–[12] for good timing performance. However, in many human-machine interaction applications, the search space for decoding speech varies dynamically depending on the user's response. A dedicated hardware architecture with a static search space has limited capabilities to deal with the dynamic nature of ASR. In addition, the architecture becomes too application-specific and targets to only ASR applications. It is unlikely that the datapath of the hardware can be reused for applications other than ASR.

As a compromise, a *hardware–software codesign* approach seems to be attractive [13], [14]. A typical hardware–software coprocessing system consists of a general purpose processor and hardware units that accelerate time critical operations to achieve required performance. Computationally intensive parts of the algorithm can be handled by the hardware accelerator(s), while sequential and control-oriented parts can be run by the processor core. The additional advantages of the hardware–software approach include the following.

1) Rapid prototyping of applications. Developers can build their applications in software without knowing every detail of the underlying hardware architecture.

2) Flexibility in design modification. The parts of the algorithm which require future modification can be implemented initially in software.

3) Universality in system architecture. The use of the general purpose processor core enables developers to integrate ASR easily with other applications.

In this paper, we present the development and tradeoffs of a hardware–software coprocessing ASR system which primarily targets on embedded applications. The system includes an optimized hardware accelerator that deals with the critical part of the ASR algorithm. The final system achieves real-time performance with a combination of software- and hardware implemented functionality and can be easily integrated into applications with voice (speech) control. The rest of this paper is organized as follows. Section II briefly describes the fundamentals of ASR. Section III gives an overview of the proposed hardware–software coprocessing recognizer. The details of the hardware accelerator and the resultant timing improvement are presented. In Section IV, an adaptive pruning scheme is proposed, which further improves the timing performance. In Section V, the proposed coprocessing system is compared with other recently reported embedded ASR systems. Finally, Section VI concludes this paper.

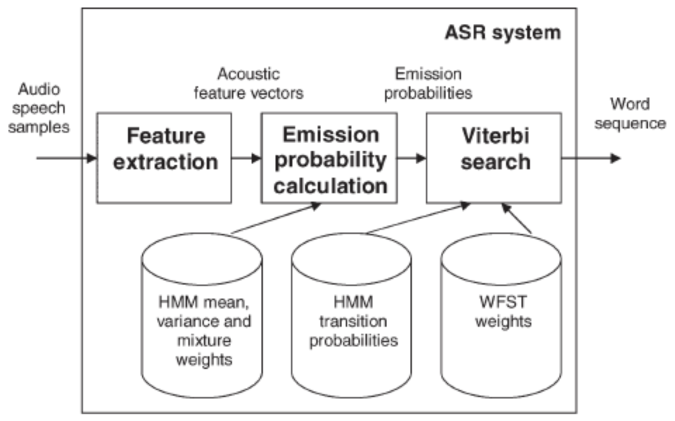

Fig. 1. Data flow diagram of a typical ASR system. The input of the system is an audio speech signal. The output is a sequence of words.

II. ASR

In a typical hidden Markov model (HMM)-based ASR system [15], three main stages are involved. Fig. 1 shows the data flow within the ASR algorithm.

The first stage is *feature extraction*. Its main purpose is to convert a speech signal into a sequence of acoustic feature vectors, $O_1^T = \{O_1, O_2 \ldots \ldots O_T\}$, where T is the number of feature vectors in the sequence. The entire speech signal is segmented into a sequence of shorter speech signals known as frames. The time duration of each frame is typically 25 ms with 15 ms of overlapping between two consecutive frames. Each frame is characterized by an acoustic feature vector consisting of D coefficients. One of the widely used acoustic features is called mel frequency cepstral coefficient (MFCC) [16]. Feature extraction continues until the end of the speech signal is reached. The next stage is the calculation of the *emission probability* which is the likelihood of observing an acoustic feature vector. The emission probability densities are often modeled by Gaussian mixture models (GMMs). The last stage is *Viterbi search* which involves searching for the most probable word transcription based on the emission probabilities and the search space.

The use of weighted finite state transducers (WFSTs) offers a tractable way for representing the search space [17]. The advantage is that the search space represented by a WFST is compact and optimal [18], [19]. Fig. 2 shows an example of a search space.

Basically, a WFST is a finite state machine with a number of states and transitions. As shown in Fig. 2, each WFST transition has an input symbol, an output symbol, and a weight. The input symbols are the triphone or biphone labels. The output symbols are the word labels. In ASR, a word is considered as a sequence of subword units called phones. Two or three phones are concatenated to form biphones or triphones. Each triphone or biphone label is modeled by an HMM. In other words, each WFST transition in Fig. 2 is substituted by an HMM. The entire WFST is essentially a network of HMM states.

The WFST weights are the language model probabilities which model the probabilistic relationship among the words in a word sequence. Usually, a word is grouped with its preceding ($n - 1$) words. The n-word sequence called n-gram is considered as a probabilistic event. The WFST weights estimate the probabilities of such events. Typical n-grams used in ASR are unigram (one word), bigram (two word, also known as wordpair grammar), and trigram (three word).

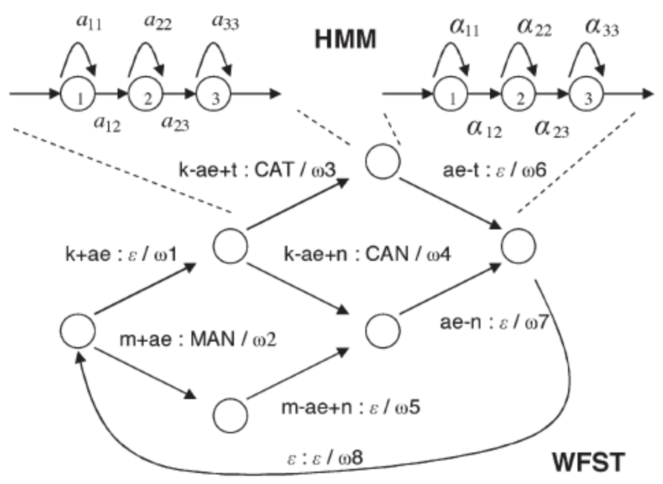

Fig. 2. Search space represented by a WFST. Each WFST transition *x* : *y/z* has three attributes. *x* is an input symbol representing a triphone or biphonelabel. *y* is an output label representing a word label. Labels can be _ which are empty labels. *z* is the weight representing a language model obability. Each triphone or biphone label is modeled by an HMM. The *aij* and *αij* are the transition probabilities of an HMM.

For implementation purposes, each HMM state has a bookkeeping entity called *token* which records the probability (*score*) of the best HMM state sequence ended at that state. Each token is propagated to its succeeding HMM states according to the topology of the search space. For example, in Fig. 2, the token in State 3 of the /k-ae+t/ HMM will replicate itself. One will propagate to its own state with HMM transition probability ($a33$ in this example) added to the token's score. Another replicated token will enter State 1 of the /ae-t/ HMM and the WFST weight ($\omega 6$) will be added to its score. When two tokens meet at an HMM state, only the better token with a higher score survives and stays at the HMM state. Other losing tokens are discarded. This method of performing the Viterbi search is known as *token passing* [20]. In addition to a score, tokens also record a sequence of word labels encountered during propagation.

The pseudocode of the ASR algorithm is shown in Fig. 3. In the beginning of the algorithm, a token is instantiated in each of HMM states at the start of each word (Line 2). Qword−start is a set of word-starting HMM states. The score of each token is reset (Line 3). After the initialization, the algorithm begins to process each frame of speech. An acoustic feature vector, **o***t*, is generated by *feature extraction* (Line 6) for each speech frame.

```
Algorithm 1 Speech recognition algorithm
1: /* $\tilde{Q}_t$ is a set of HMM states which have tokens at time $t$ */
2: $\tilde{Q}_1 \leftarrow Q_{word-start}$
3: $score_{q,1} \leftarrow 0$ for all $q \in \tilde{Q}_1$
4:
5: for $t = 1$ to $T$ do
6:    $\mathbf{o}_t \leftarrow Feature\_extraction(Frame_t)$
7:
8:    $max\_score \leftarrow \max(score_{q,t})$ for all $q \in \tilde{Q}_t$
9:    $pruning\_threshold \leftarrow max\_score - pruning\_beamwidth$
10:   $\tilde{Q}_{t+1} \leftarrow \{\}$
11:
12:   for all $q \in \tilde{Q}_t$ do
13:      if $score_{q,t} > pruning\_threshold$ then
14:         $log\_emis\_prob \leftarrow Emission\_prob\_calc(\mathbf{o}_t, q)$
15:         $\mathcal{V} \leftarrow Viterbi\_search(log\_emis\_prob, q, t)$
16:         $\tilde{Q}_{t+1} \leftarrow \tilde{Q}_{t+1} \cup \mathcal{V}$
17:      end if
18:   end for
19: end for
20:
21: $\mathcal{Q} \leftarrow \tilde{Q}_{T+1} \cap Q_{word-end}$
22: $best\_token \leftarrow \operatorname{argmax}_{q \in \mathcal{Q}}(score_{q,T+1})$
```

Fig. 3. Pseudocode of the speech recognition algorithm with beam pruning.

```
Algorithm 2   $\mathcal{V} \leftarrow Viterbi\_search(log\_emis\_prob, q, t)$
1: $\mathcal{V} \leftarrow \{\}$
2: for all $q\_suc$ states that succeed State $q$ do
3:     $new\_score \leftarrow score_{q,t} + log\_emis\_prob + trans\_weight(q, q\_suc)$
4:     if $new\_score > score_{q\_suc,t+1}$ then
5:         $score_{q\_suc,t+1} \leftarrow new\_score$
6:         $path_{q\_suc,t+1} \leftarrow path_{q,t}$
7:         $\mathcal{V} \leftarrow \mathcal{V} \cup \{q\_suc\}$
8:     end if
9: end for
10: return $\mathcal{V}$
```

Fig. 4. Pseudocode of the *V iterbi_search* function.

After feature extraction and setting the pruning threshold, the algorithm iterates through all the HMM states that have a token (Lines 12–18). If the token stays above the pruning threshold, the *emission probability* of that state is calculated (Line 14). After that, *Viterbi search* is performed on that HMM state (Line 15). Token-passing takes place during this process. It returns a set of new HMM states, *V*, which are occupied by the new tokens after token passing. The new tokens are accumulated into another set ˜*Qt*+1 which is prepared for the next speech frame.

Once all the speech frames have been processed, the best token is found among all the word-end HMM states denoted by *Q* (Lines 21–22). The best

token records its propagation path from which the word transcription can be determined.

Fig. 4 shows the pseudocode of the *V iterbi_search*() function. The for-loop iterates through all the succeeding states of *q* (Lines 2–9). For each succeeding state, *new_score* is calculated (Line 3) where the transition weight can be either the HMM transition probability for within-HMM transitions or theWFST transition weight for cross-HMM transitions. If *new_score* is reater than the score at *q_suc*, the *new_score* will update the score at *q_suc* (Line 5). The path record of the original token at *q_suc* is replaced by the path record at *q* (Line 6).

III: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 isspent 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.

System Architecture

The architecture of the hardware–software coprocessing system is shown in Fig. 5. The system consists of an Altera Nios II processor core [24] and a GMM hardware accelerator. The Nios II processor acts as the control unit of the entire system. Feature extraction and Viterbi search are implemented in software. When the system needs to perform a GMM 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) [25].

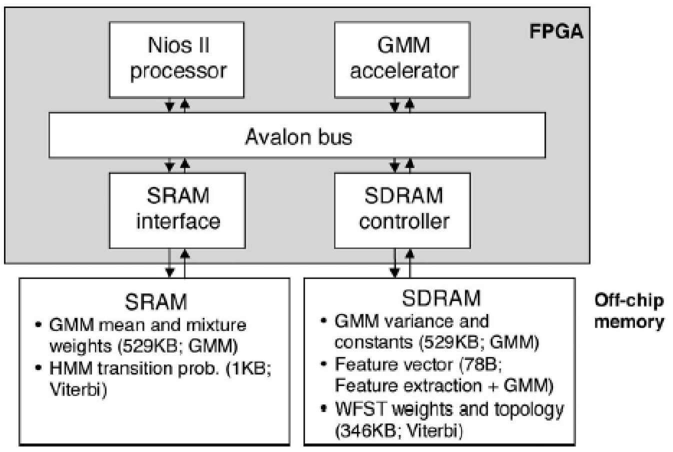
FPGA
Nios II processor
GMM accelerator
Avalon bus
SRAM interface
SDRAM controller
SRAM
• GMM mean and mixture weights (529KB; GMM)
• HMM transition prob. (1KB; Viterbi)
SDRAM
• GMM variance and constants (529KB; GMM)
• Feature vector (78B; Feature extraction + GMM)
• WFST weights and topology (346KB; Viterbi)
Off-chip memory

Fig. 5. System architecture of the hardware–software coprocessing recognizer with the GMM hardware accelerator.

Inside the brackets, it shows the data size and the ASR substages in which the data are accessed. The Nios II processor performs feature extraction and Viterbi search, while the GMM accelerator is used for GMM computation.

IV. ADAPTIVE PRUNING

In Section III, the hardware–software coprocessing system demonstrates a significant improvement on the decoding speed. 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.

A. Algorithm

Fig. 6 shows the pseudocode of the ASR algorithm with adaptive pruning. In the beginning, the beamwidth is initialized to a value (Line 4). Before token passing, the algorithm modifies the pruning beamwidth according to the number of active tokens, $n(\tilde{Q}t)$. 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 (Lines 11–12). However, if the

number of active tokens is smaller than another threshold, τlower, and also if the beamwidth is tightened previously, the beamwidth will be relaxed and its value will be increased by δ (Lines 13–16). The rest of the algorithm is the same as the one shown in Fig. 3.

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.

Another conventional pruning technique is called histogram pruning [27]. In histogram pruning, the recognizer only allows at most a certain number of active tokens to stay in the search space before the next speech frame arrives. If the number isover the allowable limit (for example, $\tilde{N}$), the $\tilde{N}$ most probable tokens remain active. Other tokens are pruned. It can guarantee a real-time factor of below 1.00 for all the utterances. However, the major issue with histogram pruning is that the recognizer will be idle in some speech frames where the number of active tokens is below the allowable limit. The idle time cannot be distributed to those speech frames which have a heavy workload (a large number of active tokens). Consequently, less probable tokens in these frames are pruned, resulting in a decrease in word accuracy.

Algorithm 3 Speech recognition algorithm with adaptive beam pruning

```
1: /* $\tilde{Q}_t$ is a set of HMM states which have tokens at time $t$ */
2: $\tilde{Q}_1 \leftarrow Q_{word-start}$
3: $score_{q,1} \leftarrow 0$ for all $q \in \tilde{Q}_1$
4: $pruning\_beamwidth \leftarrow original\_beamwidth$
5:
6: for $t = 1$ to $T$ do
7:     $\mathbf{o}_t \leftarrow Feature\_extraction(Frame_t)$
8:
9:     $max\_score \leftarrow \max(score_{q,t})$ for all $q \in \tilde{Q}_t$
10:
11:    if $n(\tilde{Q}_t) > \tau_{upper}$ then
12:        $pruning\_beamwidth \leftarrow pruning\_beamwidth - \delta$
13:    else if $n(\tilde{Q}_t) < \tau_{lower}$ then
14:        if $pruning\_beamwidth < original\_beamwidth$ then
15:            $pruning\_beamwidth \leftarrow pruning\_beamwidth + \delta$
16:        end if
17:    end if
18:
19:    $pruning\_threshold \leftarrow max\_score - pruning\_beamwidth$
20:    $\tilde{Q}_{t+1} \leftarrow \{\}$
21:
22:    for all $q \in \tilde{Q}_t$ do
23:        if $score_{q,t} > pruning\_threshold$ then
24:            $log\_emis\_prob \leftarrow Emission\_prob\_calc(\mathbf{o}_t, q)$
25:            $\mathcal{V} \leftarrow Viterbi\_search(log\_emis\_prob, q, t)$
26:            $\tilde{Q}_{t+1} \leftarrow \tilde{Q}_{t+1} \cup \mathcal{V}$
27:        end if
28:    end for
29: end for
30:
31: $\mathcal{Q} \leftarrow \tilde{Q}_{T+1} \cap Q_{word-end}$
32: $best\_token \leftarrow \operatorname{argmax}_{q \in \mathcal{Q}}(score_{q,T+1})$
```

Fig. 6. Speech recognition algorithm with adaptive beam pruning.

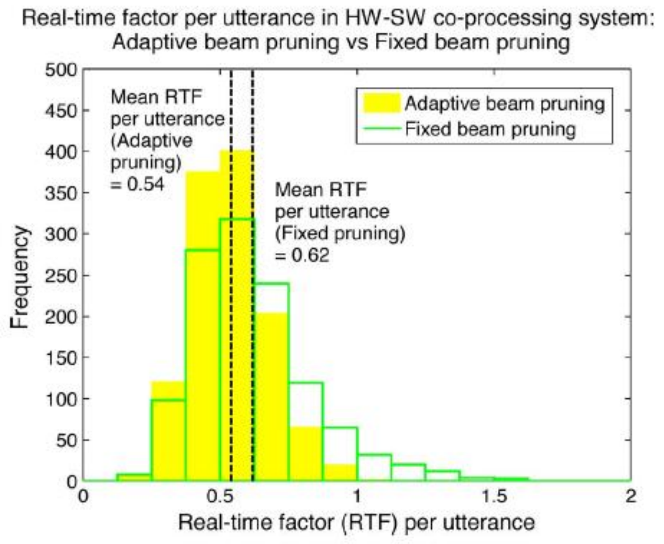

Fig. 7. Real-time factor of 1200 utterances in hardware–software coprocessing system. Adaptive beam pruning versus Fixed beam pruning.

System	CPU/Platform	Clock frequency (MHz)	Word accuracy rate (%)	Real-time factor
Pure software-based system				
PocketSphinx [8]	StrongARM	206	86.05	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	2.30
Speech Silicon [12]	Xilinx Virtex-4 XCE4VSX35 FPGA	100	N/A	N/A
Hardware-software co-processing system				
Seoul National University [13], [14]	MicroBlaze on Xilinx Virtex-4 XC4VSX35 FPGA	100	96.20[1]	N/A
Our system with fixed beam pruning	Nios II on Altera Stratix II EP2S60F672C5ES FPGA	120	93.33	0.62
Our system with adaptive beam pruning	Nios II on Altera Stratix II EP2S60F672C5ES FPGA	120	93.16	0.54

[1]The test set contains only 300 utterances.

TABLE I
PERFORMANCE OF RECENTLY DEVELOPED EMBEDDED SPEECH RECOGNITION SYSTEMS AND OUR PROPOSED SYSTEM ON THE 993-WORD RM1 TASK

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. In contrast, the adaptive pruning scheme only requires to record the number of active tokens and a few decision-making statements (if-statements) for adjusting the beamwidth once for every speech frame (Lines 11–17).

B. Timing Profile

Fig. 7 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. 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 9.75%. 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%.

V. 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 ($f$max) 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. As a result, a higher clock frequency is needed. On the other hand, pure hardwarebased systems and hardware–software coprocessing systems can run the same task at lower clock frequency.

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 [8] and In Silico Vox [10], [11] systems. The AT&T system from [9] shows slightly better word accuracy but the acoustic features are stored in files and accessed by the StrongARM platform from a PC. 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 [13], [14] 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 III, 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 [13], [14] 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.

VI. CONCLUSION

The proposed ASR system shows much better real-time factors than the other approaches without decreasing the word accuracy rate. Other advantages of the proposed approach include rapid prototyping, flexibility in design modifications, and ease of integrating ASR with other applications. These advantages, both quantitative and qualitative, suggest that the proposed coprocessing architecture is an attractive approach for embedded ASR.

The proposed GMM accelerator shows three major improvements in comparison with another coprocessing system [13], [14]. First, the proposed accelerator is about four times faster by further exploiting parallelism. Second, the proposed accelerator uses a double-buffering scheme with a smaller memory footprint, thus being more suitable for larger vocabulary tasks. Third, no assumption is made on the access pattern of the acoustic parameters, whereas the accelerator in [13] and [14] has a predetermined set of parameters.

Finally, we have presented a novel adaptive pruning algorithm which further improves the real-time factor. Compared with other conventional pruning techniques, the proposed algorithm is more flexible to deal with the time-varying number of active tokens in an utterance. 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 dditional hardware accelerators. Aside from better word accuracy and timing performance, power consumption is also another important issue for embedded applications. The proposed architecture is not tied to any specific target technology. One of the future development paths is to transfer the current implementation from FPGA to very large scale integration which consumes less power.

Our future work includes some further refinements of the speech recognition algorithm, exploration of design space to look for improvements of the hardware–software coprocessing system, and encapsulation of the speech recognizer into an intellectual property block that can be easily used in other applications.

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] M. Imai, T. Ono, and H. Ishiguro, “Physical relation and expression: Joint attention for human-robot interaction,” *IEEE Trans. Ind. Electron.*, vol. 50, no. 4, pp. 636–643, Aug. 2003.
[3] 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.
[4] H. Lam and F. Leung, “Design and training for combinational neurallogic systems,” *IEEE Trans. Ind. Electron.*, vol. 54, no. 1, pp. 612–619, Feb. 2007.
[5] 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.
[6] N. Hataoka, Y. Obuchi, T. Mitamura, and E. Nyberg, “Robust speech dialog interface for car telematics service,” in *Proc. IEEE Consumer Commun. Netw. Conf.*, 2004, pp. 331–335.
[7] 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.
[8] D. Huggins-Daines, M. Kumar, A. Chan, A. Black, M. Ravishankar, and A. Rudnicky, “Pocketsphinx: A free, real-time continuous speech recognition system for hand-held devices,” in *Proc. ICASSP*, 2006, pp. 185–188.
[9] E. Bocchieri and D. Blewett, “A decoder for LVCSR based on fixed-point arithmetic,” in *Proc. ICASSP*, 2006, pp. 1113–1116.
[10] 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.
[11] E. Lin, K. Yu, R. Rutenbar, and T. Chen, “A 1000-word vocabulary, speaker-independent, continuous livemode speech recognizer implemented in a single FPGA,” in *Proc. Int. Symp. Field-Programmable Gate Arrays*, 2007, pp. 60–68.
[12] J. Schuster, K. Gupta, R. Hoare, and A. Jones, “Speech silicon: An FPGA architecture for real-time, hidden Markov model based speech recognition,” *EURASIP J. Embedded Syst.*, vol. 2006, no. 1, pp. 1–19, Jan. 2006.
[13] 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.

[14] K. You, H. Lim, and W. Sung, “Architectural design and implementation of an FPGA softcore based speech recognition system,” in *Proc. 6th Int. Workshop Syst. Chip Real Time Appl.*, 2006, pp. 50–55.
[15] L. Rabiner, “A tutorial on hidden Markov models and selected applications in speech recognition,” *Proc. IEEE*, vol. 77, no. 2, pp. 257–286, Feb. 1989.

[16] S. Davis and P. Mermelstein, “Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences,” *IEEE Trans. Acoust., Speech, Signal Process.*, vol. ASSP-28, no. 4, pp. 357–366, Aug. 1980.
[17] M. Mohri, “Finite-state transducers in language and speech processing,” *Comput. Linguist.*, vol. 23, no. 2, pp. 269–311, Jun. 1997.
[18] F. Pereira and M. Riley, “Speech recognition by composition of weighted finite automata,” in *Finite-State Language Processing*. Cambridge, MA: MIT Press, 1997, pp. 431–453.
[19] M. Mohri, “Weighted finite-state transducer algorithms: An overview,” in *Formal Languages and Applications*, vol. 148. Heidelberg, Germany: Physica-Verlag, 2004, pp. 551–564.
[20] S. Young, N. Russell, and J. Thornton, *Token Passing: A Simple Conceptual Model for Connected Speech Recognition Systems*. Cambridge, U.K.: Eng. Dept., Cambridge Univ., 1989.
[21] H. Ney, D. Mergel, A. Noll, and A. Paeseler, “A data-driven organization of the dynamic programming beam search for continuous speech recognition,” in *Proc. ICASSP*, 1987, pp. 833–836.
[22] H. Ney, R. Haeb-Umbach, B. Tran, and M. Oerder, “Improvements in beam search for 10000-word continuous speech recognition,” in *Proc. ICASSP*, 1992, pp. 9–12.
[23] S. Young, G. Evermann, M. Gales, T. Hain, D. Kershaw, X. Liu, G.Moore, J. Odell, D. Ollason, D. Povey, V. Valtchevnt, and P. Woodland, *The HTK Book (for HTK Version 3.4)*. Cambridge, U.K.: Eng. Dept., Cambridge Univ., 2006.
[24] *Nios II Processor Reference Handbook*, Altera Corp., San Jose, CA, 2006.
[25] *Nios Development Board Reference Manual, Stratix II Edition*, Altera Corp., San Jose, CA, 2005.
[26] P. Price, W. Fisher, J. Bernstein, and D. Pallett, “The DARPA 1000-word resource management database for continuous speech recognition,” in *Proc. ICASSP*, 1988, pp. 651–654.
[27] V. Steinbiss, B. Tran, and H. Ney, “Improvements in beam se search,” in *Proc. ICSLP*, 1994, pp. 2143–2146.