Real-Time Continuous Speech Recognition System on SH-4A Microprocessor

Hiroaki Kokubo*, Nobuo Hataoka*, Akinobu Lee†, Tatsuya Kawahara‡ and Kiyohiro Shikano§
*Central Research Laboratory, Hitachi Ltd, Tokyo, Japan.
Email: hiroaki.kokubo.dz@hitachi.com, hataoka@tohtech.ac.jp
†Nagoya Institute of Technology, Nagoya, Japan. Email: ri@nitech.ac.jp
‡Kyoto University, Kyoto, Japan. Email: kawahara@i.kyoto-u.ac.jp
§Nara Institute of Science and Technology, Nara, Japan. Email: sikano@is.naist.jp

***Abstract*—To expand CSR (continuous speech recognition) software to the mobile environmental use, we have developed embedded version of Julius (embedded Julius). Julius is open source CSR software, and has been used by many researchers and developers in Japan as a standard decoder on PCs. In this paper, we describe an implementation of the embedded Julius on a SH-4A microprocessor. SH-4A is a high-end 32-bit MPU (720MIPS) with on-chip FPU. However, further computational reduction is necessary for the embedded Julius to operate real-time. Applying some optimizations, the embedded Julius achieves real-time processing on the SH-4A. The experimental results show 0.89 x RT(real-time), resulting 4.0 times faster than baseline CSR. We also evaluated the embedded Julius on large vocabulary (20,000 words). It shows almost real-time processing (1.25 x RT).**

I. Introduction

We have developed embedded CSR software named "embedded Julius" on a SH-4 microprocessor (Renesas Technology Corp.[2])[3]. Julius[4] is open source CSR software developed on PCs. It has been used by many researchers and developers in Japan as a standard decoder. The largest advantage of using Julius is that Julius supports standard formats of language/acoustic models. Developers are free to use acoustic/language models created by popular modeling tools (such as HTK[5], CMU-Cambridge Toolkit[6]). In previous work[3], we reported processing speed of the embedded Julius was 2.23 x RT on the condition of 5,000 words vocabulary. Our goal is to realize the embedded Julius operating at real-time on a microprocessor.

Recently, some studies on CSR systems for portable devices were reported. Ishikawa et. al. [7] developed 50,000 words CSR on multi-core CPU (ARM9 core (150MHz) x 3). Generally speaking, multi-core CPU has high performance and enables to handle large vocabulary. On the other hand, coding of CSR deeply depends on CPU architecture. For single-core processors, Huggins-Daines et. al.[8] released POCKET SPHINX, which operates real-time on Sharp Zaurus (206MHz Strong ARM, 235MIPS). Because Strong ARM processor has no hardware support for floating point operations, vocabulary size was only 1,000 words and word error rate was degraded to the baseline system by cost reduction schemes (ex. fixed point operations). Our target microprocessors are SH-4 series[2], which are high-end processors of SH family. A SH-4 core includes a high-speed FPU. Restrictions on CPU performance are few in comparison with Strong ARMs.

In this paper, we report a development of the embedded Julius for the SH-4A microprocessor, which is upper version of a SH-4. However operating frequency of the SH-4A is 1.67 times faster than the SH-4, further computational reduction is necessary for the embedded Julius to operate in real-time. We also describe some optimizations to realize real-time processing.

II. System overview

A. SH-4A microprocessor

The SH-4A[2] is a RISC processor with operating frequency of 400MHz. The instruction set is fully SH-4 upward compatible. It realizes a processing performance of 720 MIPS. Furthermore, the SH-4A core includes an FPU that supports both single-precision and double-precision arithmetic operations. The four-way set-associative cache memory is divided into two 32-kbyte areas, one for instructions and one for data.

B. Developmental hardware platform

A T-engine[9] board is a developmental hardware platform which has common operating system (OS) called eTRON. Table I shows specifications of the T-engine board. Fig. 1 and Fig. 2 show a photo and architecture of the T-engine board respectively. The T-engine platform consists of a CPU board, an LCD board. The CPU board has one microprocessor(SH-4A), 128MB of work memory and many interfaces. Program, dictionary, language models and acoustic models are stored in a CF (Compact Flash) card. Speech data is digitized at

TABLE I
Specification of T-engine Board

CPU	SH-4A(SH7780) (720 MIPS, 2.8 GFLOPS)
Operating Freq.	Internal: 400MHz External: 100MHz
User RAM	128MByte
OS	T-Kernel
Audio CODEC (A/D)	16kHz,16bit
LDC	TFT color, 240 x 320
Size / Power	120 mm x 75 mm / DC 5.6V

1-4244-1274-9/07/$25.00 ©2007 IEEE

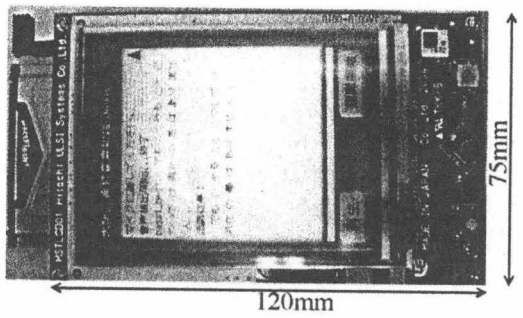

Fig. 1. T-engine Board

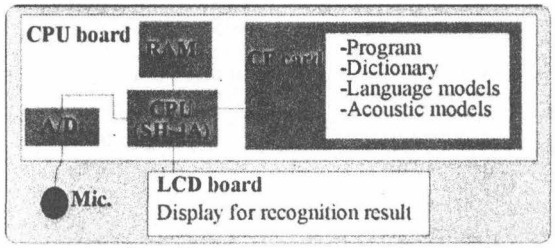

Fig. 2. Architecture of T-engine Board

16kHz/16bit sampling on Audio codec. Recognition result displays on an LCD.

C. Baseline CSR system

Julius[4] is a high-performance, two-pass large vocabulary continuous speech recognition software. Julius is distributed with open license together with source codes, and has been used by many researchers and developers in Japan. Major search techniques are fully incorporated such as tree lexicon, N-gram factoring, cross-word context dependency handling, enveloped beam search, Gaussian pruning, Gaussian selection, etc. Standard formats are supported to cope with a popular modeling toolkit. Based on word trigram and context-dependent HMM, it can perform almost real-time decoding on most current PCs in 20,000 words dictation task.

In order to distinguish Julius from the embedded Julius, we call the baseline CSR a conventional Julius.

III. Optimal implementation

Through a development tool-kit for the T-Engine, Porting Julius to the SH-4A was done without much difficulty. Source codes of Julius written in C language can be compiled without substantial modifications, excluding I/O depended modules (audio modules, touch panel modules, etc.). Most difficulty was computational reduction. Processing speed of the conventional Julius was far from real-time on the SH-4A. In this section, we describe optimizations for real-time processing.

A. Avoid data access from external device

In previous version of the embedded Julius worked on SH-4, Language models were read from an external device (CF card), because the SH-4 T-engine has only 64MB RAM. I/O speed of the CF card is much slower than that of RAM. For the SH-4A T-engine, which has enough user memory (128MB RAM), Language models are loaded to RAM at start-up to avoid overhead of data access from external device.

B. Reduce memory fragmentation

In decoding process, a lot of word hypotheses are generated and some of them are pruned. System functions (memory allocation / free) are frequently called at each time. Overhead of these system calls is negligible on PC, but these can not be ignored on T-engine. Memory fragmentation gets serious especially in case of recognizing long utterance. To avoid memory fragmentation, memory management for word hypotheses was changed as follows.

Memory area for hypotheses is secured in advance at initial set-up. This memory area is partitioned at fixed interval which is equivalent to data size of each word hypothesis. Then, memory manager can reallocate a new hypothesis to the memory pointer where hypothesis was already pruned.

IV. Gaussian mixture selection

A major part of computational cost for CSR is calculation of pdf (probability density function). To realize CSR software with low calculation cost, computational reduction for pdf calculations is needed. The conventional Julius already introduces GMS (Gaussian Mixture Selection) algorithm[12], which reduces the cost of pdf calculations. GMS is effective for reducing computational cost of pdf calculations, but almost half of the total process time is still taken by pdf calculations. Additional process reduction is needed.

We have proposed a modified GMS method[3]. In this section, we describe a conventional GMS briefly, and then explain its modifications.

A. Conventional GMS

The GMS method[12] is a procedure to select Gaussian distributions by the HMM states using a hierarchical relationship between monophone models and triphone models. Fig. 3 shows a GMS procedure. For each frame, all Gaussians of monophone HMM states are computed for preliminary evaluation. Only HMM states corresponding to k-best states on preliminary evaluation are selected for computations of triphone models. Others are assigned to scores calculated on monophone models.

B. Further cost reduction

We proposed two modifications on GMS as follows[3].

1) Computational reduction on mixture selection stage: In the conventional GMS method, all HMM states in monophone models are calculated for the mixture selection, since information of active states is not available at that time. Knowing where hypotheses stay alive at time $t-1$, only HMM states which these hypotheses may be visited are calculated for the mixture selection stage at time t. Applying this modification, computational cost is reduced for the mixture selection compared to the conventional GMS method, which calculates all

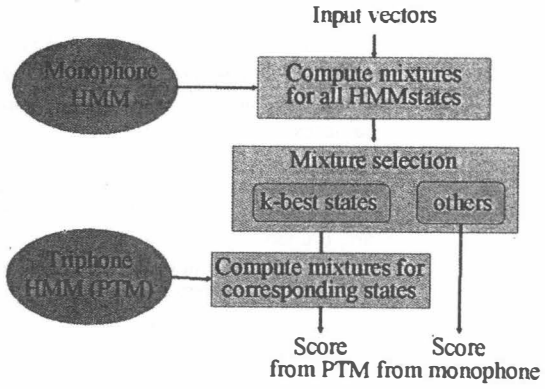

Fig. 3. Conventional Gaussian Mixture Selection (GMS)

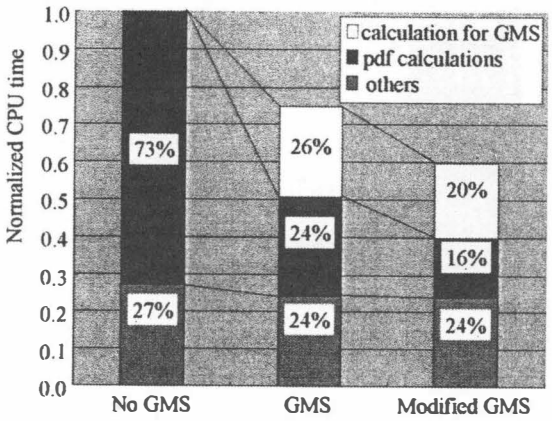

Fig. 4. Profile of Computational Cost Measured on PC

states in monophone models. Furthermore, since there is no fear of selecting a useless state whose pdf score is not needed to calculate, a small number of k-best states could be specified without any degradation of recognition performance.

2) Gaussian selection within HMM state: Gaussian selection (GS)[10] is based on the idea "only Gaussians close to an input vector have dominant effect on a pdf score of an HMM state". On the other hand, the conventional GMS method calculates all Gaussians within an HMM state, even though a pdf score derived from Gaussian distant from the input vector is negligible. For reducing pdf calculations on HMM states, the calculation strategy is changed: calculating only Gaussians neighboring the input vector, instead of calculating all Gaussians. By this procedure the computation cost is much more reduced.

C. Efficiency

In previous work[3], efficiency of the GMS method was evaluated on PC simulation. Fig. 4 shows computational cost of Julius measured on a PC. The vertical axis is normalized CPU time calculated by Linux command "gprof". In no GMS case, 73% of total computational cost was taken by pdf calculations. Adopting the conventional GMS method, the total computational cost was reduced to 74% of no GMS in spite of additional costs for calculating monophone models. Modifying the GMS method (the proposed GMS), the normalized CPU time for GMS was reduced 26% to 20%; the normalized CPU time of pdf calculations was reduced 24% to 16%. As a result, the total computational cost was reduced to 60% of no GMS.

V. Experimental Results and Discussions

A. experimental setup

We evaluated system performance of the embedded Julius on the SH-4A T-engine platform. Table II shows experimental conditions. The vocabulary size was 5,000 words. Julius is a two-pass decoder using a bigram language model for the 1st-pass and a trigram language model for the 2nd-pass. Acoustic models were PTM[11] having 3,000 states with 64 mixtures. Monophone models with 129 states and 16 mixtures/states were used for the GMS. Test-set speech was selected 60 sentences (30 male, 30 female) from JNAS (Japanese Newspaper Article Sentences) corpus[13].

B. evaluation of optimizations

Experimental results are shown in Table III. We evaluated in three optimum conditions: (1) without optimization (Non), (2) Avoid data access from CF card (CF), (3) reduce memory fragmentation (Frag.) and three GMS conditions: (1) without GMS (Non), (2) Conventional GMS (Conv.), (3) Modified GMS (Modify).

Memory size consumed on the embedded Julius was about 50 MB. Adopting the GMS algorithm, additional memory space was needed for monophone models used on the mixture selection. Word accuracies of 5 conditions were almost the same. Optimizations of implementation did not change word accuracies (condition #1-#3). The GMS method, which simplified pdf calculations, affected recognition performance slightly. This result is in consistency with previous work[3]. Regarding recognition speed, the embedded Julius on the condition #1 (without optimizations, without GMS) operated 3.57 x RT. By optimal implementation, recognition speed was reduced to 1.21 x RT (condition #3). By combining the optimization and the modified GMS method (condition #5), the embedded Julius finally achieved real-time processing (0.89 x RT).

TABLE II
Experimental Conditions

Vocabulary size	5,000 words
Acoustic models	PTM (3,000 states, 64 mixtures)
Monophone HMM (used for GMS)	129 states, 16 mixtures
Language models	bigram (for 1st pass) trigram (for 2nd pass)
Test speech	60 sentences in JNAS corpus[13]

TABLE III
PERFORMANCE ON VARIOUS OPTIMUM CONDITIONS

	Optimize	GMS	Memory size	Word accuracy	x RT
#1	Non	Non	49.6MB	89.1%	3.57
#2	CF	Non	50.1MB	89.1%	1.45
#3	CF+Frag.	Non	50.1MB	89.1%	1.21
#4	CF+Frag.	Conv.	51.0MB	89.3%	1.11
#5	CF+Frag.	Modify	51.0MB	89.7%	0.89

TABLE IV
PROFILE OF LANGUAGE MODELS

Vocabulary size	bigram entry	trigram entry
5,000	787,365	453,446
20,000	1,675,804	744,438

C. evaluation on large vocabulary condition

We also evaluated the embedded Julius on a large vocabulary condition (20,000 words). In this experiment, a dictionary size was expanded 5,000 words to 20,000 words. Table IV shows a profile of language models. Based on entropy criteria, trigram entries for 20k language model were compressed by 10% [14]. Acoustic models are same as 5,000 words dictation task. Pruning parameter was set to appropriate value in order to vocabulary size.

Experimental result is shown in TableV. Memory size was increased 51.0MB to 94.1MB in proportion to vocabulary size. Word accuracy in condition with the modified GMS was declined by 1%, but it was not serious damage. Processing speed was 1.60 x RT for no GMS and 1.25 x RT for the modified GMS respectively. Applying the modified GMS, recognition speed was 1.25 x RT, which was 1.3 times faster than that of no GMS.

D. Discussions

In this section, we evaluated the embedded Julius on the SH-4A T-engine. Experimental results indicated that memory requirement for the embedded Julius was 51 MB on 5,000 words dictation task and 94.1 MB on 20,000 words dictation task. It isn't big problem for mobile applications, because commercial PDAs have 64-256MB of work memory. Regarding processing speed on vocabulary size of 5,000 words, the embedded Julius achieved real-time processing (0.89 x RT) on the SH-4A (720MIPS). On the condition of large vocabulary (20,000 words), the embedded Julius showed 1.25 x RT.

TABLE V
PERFORMANCE ON LARGE VOCABUALY TASK (20,000 WORDS)

	Optimize	GMS	Memory size	word accuracy	x RT
#1	CF+Frag.	Non	93.0MB	91.8%	1.60
#2	CF+Frag.	Conv.	94.1MB	92.0%	1.44
#3	CF+Frag.	Modify	94.1MB	90.9%	1.25

VI. CONCLUSIONS

This paper describes embedded Julius on a SH-4A T-engine. The optimal implementation and the modified GMS method are applied for computational reduction. This approach does not change the structure of acoustic models in consistency with that used by the conventional Julius, and enable developers to use acoustic models created by popular modeling tools. The experimental result shows 0.89 times of RT, resulting 1.4 times faster than that of no GMS and 4.0 times faster than that without any optimizations. We also show almost real-time processing (1.25 x RT) on large vocabulary task (20,000 words).

In future, we will investigate more compact and more noise robust. For the noise robustness, we are developing a noise reduction front-end. We will also develop an application prototype using the embedded Julius on a mobile platform.

ACKNOWLEDGMENT

This research activity has been supported by the e-society project founded by Ministry of Education, Culture, Sports, Science and Technology, Japan.

REFERENCES

[1] N. Hataoka, K. Kokubo, Y. Obuchi, and A. Amano, "Development of robust speech recognition middleware on microprocessor," ICASSP, pp.837-840, 1998.
[2] http://www.renesas.com/
[3] H.Kokubo, N.Hataoka, A.Lee, T.Kawahara, and K.Shikano, "Embedded Julius: Continuous Speech Recognition Software for Microprocessor," MMSP, pp.378-381, 2006.
[4] A.Lee, T.Kawahara, and S.Doshita, "An efficient two-pass search algorithm using word trellis index," ICSLP, pp.1831-1834, 1998.
[5] S.Yang, G.Evermann, T.Hain, D.Kershaw, G.Moore, J.Odell, D.Ollanson, D.Povey, V.Valtchev, and P.Woodland, The HTK book(for HTK version 3.2.1), In Cambridge University Engineering Department, 2002.
[6] P.R.Clarkson and R.Rosenfeld, "Statistical language modeling using the CMU-Cambridge toolkit," Eurospeech, vol.5, pp.2707-2710, 1997.
[7] S.Ishikawa, K.Yamabane, R.Isotani, A.Okumura,"Parallel LVCSR algorithm for cellphone-oriented multicore processors," ICASSP, vol.I, pp.177-180, 2006.
[8] D.Huggins-Daines, M.Kumar, A.Chan, A.Black, M.Ravishankar, A.Rudnicky, "POCKETSPHINX : A Free Real-time continuous speech recognition system for hand-held devices," ICASSP, vol. I, pp.185-188, 2006.
[9] http://www.t-engine.org/index.html
[10] E.Bocchieri, "Vector quantization for efficient computation of continuous density likelihoods," ICASSP, pp.692-695, 1993.
[11] A.Lee, T.Kawahara, K.Takeda, and K.Shikano, "A new phonetic tied-mixture model for efficient decoding," ICASSP, pp.1269-1272, 2000.
[12] A.Lee, T.Kawahara, and K.Shikano, "Gaussian mixture selection using context-independent HMM," ICASSP, pp.69-72, 2001.
[13] K.Itoh, M.Yamamoto, K.Takezawa, T.Matsuoka, K.Shikano, T.Kobayashi, and S.Itahashi, "The design of the newspaper-based Japanese large vocabulary continuous speech recognition corpus," ICSLP, pp.3261-3264, 1998.
[14] N.Yodo, K.Shikano and S.Nakamura,"Compression Algorithm of Trigram Language Models based on Maximum Likelihood Estimation," ICSLP, pp.1684-1686, 1998.