Kobe University Repository : Kernel

PDF issue: 2024-03-29

VLSI Architecture of GMM Processing and Viterbi Decoder for 60,000-Word Real-Time Continuous Speech Recognition

Noguchi, Hiroki ; Miura, Kazuo ; Fujinaga, Tsuyoshi ; Sugahara, Takanobu ; Kawaguchi, Hiroshi ; Yoshimoto, Masahiko

(Citation)
IEICE Transactions on Electronics,94(4):458-467

(Issue Date)
2011-04-01

(Resource Type)
journal article

(Version)
Version of Record

(Rights)
copyright©2011 IEICE

(URL)
https://hdl.handle.net/20.500.14094/90002965

KOBE UNIVERSITY

IEICE TRANS. ELECTRON., VOL.E94–C, NO.4 APRIL 2011

PAPER Special Section on Circuits and Design Techniques for Advanced Large Scale Integration

VLSI Architecture of GMM Processing and Viterbi Decoder for 60,000-Word Real-Time Continuous Speech Recognition*

Hiroki NOGUCHI[†a)], *Student Member*, Kazuo MIURA[†], Tsuyoshi FUJINAGA[†], Takanobu SUGAHARA[†], *Nonmembers*, Hiroshi KAWAGUCHI[†], *and* Masahiko YOSHIMOTO[†], *Members*

SUMMARY We propose a low-memory-bandwidth, high-efficiency VLSI architecture for 60-k word real-time continuous speech recognition. Our architecture includes a cache architecture using the locality of speech recognition, beam pruning using a dynamic threshold, two-stage language model searching, a parallel Gaussian Mixture Model (GMM) architecture based on the mixture level and frame level, a parallel Viterbi architecture, and pipeline operation between Viterbi transition and GMM processing. Results show that our architecture achieves 88.24% required frequency reduction (66.74 MHz) and 84.04% memory bandwidth reduction (549.91 MB/s) for real-time 60-k word continuous speech recognition.

key words: *speech recognition, hidden Markov model (HMM), VLSI architecture*

1. Introduction

Speech recognition technology has been used recently in various applications such as cellular telephones, car-navigation systems, PDAs, wearable computers, and robotics. Nevertheless, the large vocabulary — more than 60-k words — real-time continuous speech recognition (LVRCSR) with an accurate model is too resource-hungry and power-sensitive for use in software applications [3].

A hardware approach, with implementation by VLSI or an FPGA, can achieve more compact and more battery-friendly speech recognition because of its advantageous processing speed and power consumption. To enhance speech-recognition performance, some studies have applied hardware approaches. Lin et al. investigated FPGA implementations for 5-k word continuous speech recognition [4], [5], but the applications did not run in real time. Choi et al. investigated FPGA implementations for 20-k word speech recognition [6], [7], but both consumed slightly higher memory bandwidth (BW) and power. Ma et al. reported memory-bandwidth reduction of Gaussian Mixture Models (GMM) processing for real-time 20-k word speech recognition [8], but that method did not treat Viterbi processing. Therefore, memory bandwidth reduction on a Viterbi processor remains as an important task because it requires high memory bandwidth.

A comparison of external memory bandwidth among recently described hardware-based speech recognizers is shown in Fig. 1. To date, the hardware approach has never achieved real-time operation with a 60-k word language model because numerous computations and external memory bandwidth degrade as the vocabulary is increased. In the future ubiquitous computing era, with further development of robotics technology, speech recognition systems are expected to become the main technique for human interface devices for mobile, wearable, and intelligent robots. Actually, LVRCSR is expected to become a key technology for such applications.

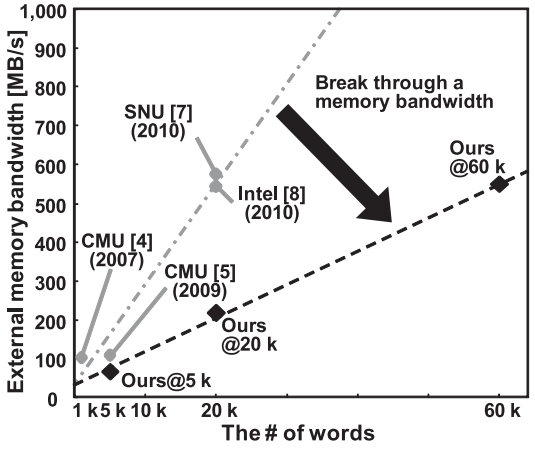


Fig. 1 Conventional architectures for real-time speech recognition and our target performance.

As described in this paper, we propose a novel architecture to reduce the required computational cycle time and memory bandwidth. Our architecture comprises specialized cache, threshold-cut beam pruning, two-stage language model search, parallel processing, and pipeline operation between Viterbi transition and GMM processing. Using that architecture, high-efficiency and low memory-bandwidth Viterbi and GMM processing can be implemented. Thereby, more sophisticated LVRCSR can be applied to VLSI.

The remainder of this paper is organized as follows. Section 2 introduces the theory and algorithms of speech recognition with the HMM algorithm. Section 3 presents novel architectural techniques for GMM and Viterbi processors. Section 4 specifically describes the GMM and Viterbi architecture. Section 5 presents an assessment of the performance of the architecture. Finally, Sect. 6 summarizes this paper.

Manuscript received August 14, 2010.
Manuscript revised November 9, 2010.
[†]The authors are with Kobe University, Kobe-shi, 657-8501 Japan.
*The preliminary version was presented at [1], [2].
a) E-mail: h-nog@cs28.cs.kobe-u.ac.jp
DOI: 10.1587/transele.E94.C.458

Copyright © 2011 The Institute of Electronics, Information and Communication Engineers

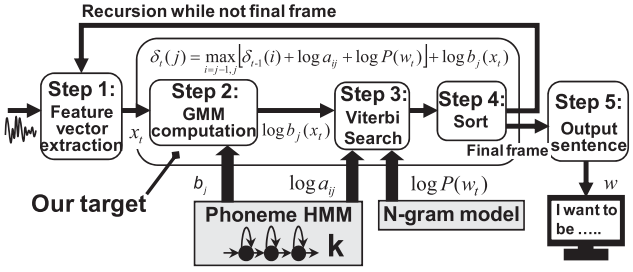


Fig. 2 Speech recognition flow with the HMM algorithm.

2. Speech Recognition Overview

Figure 2 presents the speech recognition flow with the HMM algorithm. The following items describe concrete stages. Step 1: Feature vector extraction: a feature vector is extracted on a frame-by-frame basis. Step 2: GMM calculation: a phonemic-model GMM is read and GMM probability, $\log[b_j(x_t)]$, is calculated for all active state nodes. Step 3: Viterbi transition: $\delta_t(j)$ is calculated for all active state nodes using GMM probabilities. Step 4: Beam pruning: according to the beam width, active state nodes having a higher score (accumulated probability) are selected; the others are dumped. Step 5: Output sentence: The word-end state and having the maximum score is output as a speech recognition result after final-frame calculation and determination of the transition sequence.

2.1 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 Eq. (1), $\log[b_j(x_t)]$ is expressed as a logarithm of a sum of the Gaussian distribution multiplied by weight functions. We assume that Σ_i is a diagonal matrix and simplify it.

$$\begin{aligned}\log b_j(x_t) &= \sum_i^{mix} \lambda_i N(x_t, \mu_i, \Sigma_i) \\ &= \log\left[\sum_i^{mix} \lambda_i \left[\frac{1}{(2\pi)^{\frac{P}{2}} |\Sigma_i|^{\frac{1}{2}}} \exp\left\{-\frac{1}{2}(x_t-\mu_i)^t \Sigma_i (x_t-\mu_i)\right\}\right]\right] \\ &= add\ \log\left[w_{ij} + \sum_{s=1}^{P} (x_{ts}-\mu_{ijs})^2 \sigma_{ijs}\right] \qquad (1)\end{aligned}$$

$$w_j = \log\lambda_i + \log\left[\left((2\pi)^{\frac{P}{2}} \left(\prod_{s=1}^{P} \Sigma_{ijs}\right)^{\frac{1}{2}}\right)^{-1}\right],$$

$$add\ \log[X_i] = \log \sum_{i=1}^{mix} X_i \qquad (2)$$

In those equations, the following parameters are used: $b_j(x_t)$ is a GMM probability density function (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. Also, w_{ij} is a constant number that can be computed offline before speech recognition. Equation (2) shows that the GMM computation at one dimension consists of one addition, one subtraction, two multiplications, P summations, and their respective logarithms.

2.2 Time-Synchronous Viterbi Beam Search

The following formulas show the log-Viterbi algorithm. To prevent underflow, logarithms are usually taken.
Initialization:

$$\delta_0(0) = \log\pi \qquad (3)$$

Recursion:

$$\begin{aligned}&\delta_t(j) = \max_{i=j-1,\ j} [\delta_{t-1}(i) + \log a_{ij}] + \log b_j(x_t) \\ &\text{for } 1 \le t \le T,\ 1 \le j \le N_{state} \qquad (4)\end{aligned}$$

Termination:

$$P(w|x_1, x_2, \cdots, x_T) = \max_{N_f}[\delta_T(i)] \qquad (5)$$

Therein, T represents the number of frames, N_{state} denotes the number of all HMM states, N_f stands for the states set that correspond to word-end, and i and j are state indexes. In addition, $\delta_t(j)$ is a likelihood value at a time index t and state j; w is a recognition output sentence.

The speech wave form is divided into frames (15–25 ms); a feature vector is calculated in each frame. Equation (4) 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.

In reality speech recognition, N_{state} is from 1,000 to 5,000 states. It is too large to calculate all likelihood values. To address this problem, after all transitions in one frame are over, only a few (hereinafter, we call it the "beam width") nodes with large likelihood values are considered. The remaining nodes are terminated. We designate this process as *beam pruning*. Nodes that are unpruned in later stages are designated as active state nodes. In the next frame, only the likelihood values in the active state nodes are computed. After the final frame of computation, the maximum likelihood value in the word-end state is output as the recognition result.

2.3 Referential Hardware Design

We implemented referential hardware of a speech recognition system based on the Julius 4.0 [3], a well-known Japanese speech recognition system software by using Verilog hardware description language (HDL) [1], [2]. This architecture comprised of a GMM processor and a Viterbi processor. The GMM processor adopts a four-core parallelism,

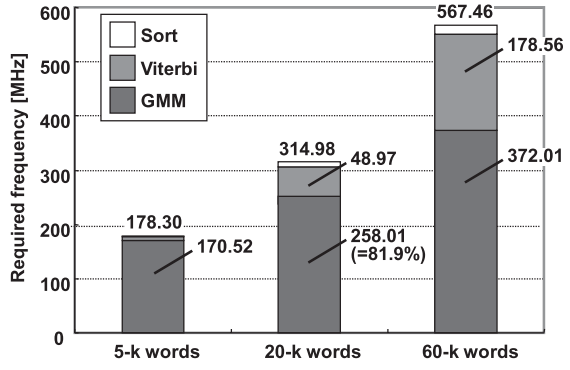


Fig. 3 Required frequency in a real-time process with the referential hardware [1], [2].

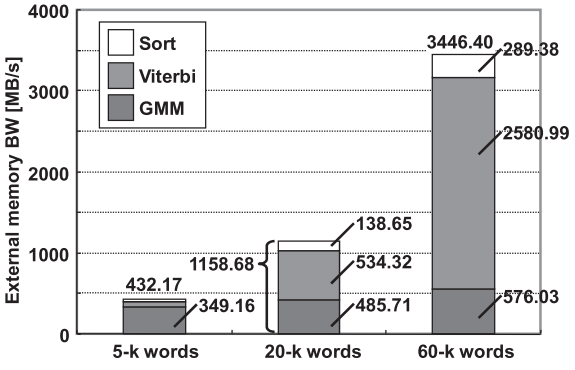


Fig. 4 Required memory bandwidth in a real-time process with the referential hardware [1], [2].

pipelined processing, and a vector look-ahead scheme [1]. In the vector look-ahead scheme, several feature vectors are buffered in advance, and their output probabilities are computed in parallel. Then, answers are stored in a cache. If a duplicated state appears at a following frame, the answer stored in the cache is read out. The Viterbi processor adopts a dual-core parallelism and asynchronous scheduling to reduce the required minimum operating frequency and the idle cycles [2].

2.4 Computation Amounts and Memory Bandwidth

We first profiled the referential hardware (mentioned in Sect. 2.3) of speech recognition system, using 5-k, 20-k, and 60-k word speech recognition models. The beam widths were, respectively, set to 500, 2,000, and 4,000 with 5-k, 20-k, and 60-k word models. We utilized an FPGA (Stratix *II*; Altera Corp.) to obtain the required frequencies and the memory bandwidths needed for the three models.

Figures 3 and 4 respectively show the required frequency and the required memory bandwidth in the referential hardware to achieve real-time speech recognition. The GMM processing dominates a large portion of the total computation time, of which the computing output probabilities occupy 81.9% considering a 20-k word LVRCSR. For LVRCSR recognition, minimizing the computation time of the output probabilities is effective in terms of the computational workload. In contrast, Viterbi search consumes a larger share of the total memory bandwidth as the number of words increases. When developing a VLSI chip for LVRCSR, a salient issue is the high memory bandwidth of speech recognition processing, which engenders inefficient power consumption. It is necessary to reduce the memory bandwidth to develop a low-power VLSI chip for use with LVRCSR. Figure 4 shows that, when considering a 20-k word LVRCSR, memory bandwidths of the Viterbi, GMM, and sort processing are estimated respectively as 534.32 MB/s, 485.71 MB/s, and 138.65 MB/s. Furthermore, when considering a 60-k word LVRCSR, the memory bandwidth of Viterbi search increases by 483% (2580.99 MB/s) compared to a 20-k word LVRCSR. As described in this paper, to realize a low-power and low-memory-bandwidth 60-k word recognition system, we propose several ideas to reduce the workload and memory bandwidth.

3. Proposed Schemes

3.1 Burst GMM Calculation

The memory bandwidth in GMM calculations results from large-sized GMM parameters. Each state in the HMM has a specific GMM. To compute the likelihood $\log[b_j(x_t)]$, the memory controller must read the Gaussian parameters, the mean μ_{ijs} and the standard deviation σ_{ijs}, from external memory. However, each phonemic HMM has a self-transition; 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%. To reduce the external memory bandwidth for reading the acoustic model, we share the Gaussian parameters to compute GMM probability among several contiguous frames at a time [1]. If we were to share the Gaussian parameters for 50 frames, then we would need to increase the GMM result RAM used for storing the calculated scores, and the input buffer, which is stored the MFCC feature vectors, from 15 kB to 750 kB and from 200 B to 10 kB. However, the external memory bandwidth can be reduced to 1/50 if all Gaussian parameters can be shared. For 20-k and 60-k word recognition, it is necessary to maintain sufficient beam width according to the number of words to achieve highly accurate recognition. Furthermore, it engenders a large amount of GMM processing among approximately all GMM states. In our novel GMM architecture, for all input feature vectors, every GMM probability is computed. In doing so, a two-stage pipeline between GMM and Viterbi can be applied easily.

3.2 Modified Unigram Language Model

A unigram language model was used for computing word-internal transitions. The unigram language model comprises HMM probabilities; each value corresponds individually to a state of HMM trees. In the conventional scheme, if the HMM state transitioned, then the probability of the previous

state subtracted from the temporal score before being added the new probability of the current state to the temporal score. In terms of a unigram language model, the previous state of every state is identifiable individually. For that reason, we modified the unigram language model to hold only difference values between the probability of a new HMM state and the probability of its previous HMM state. Using our modified unigram language model, the extra memory access to the previous state can be reduced. Furthermore, because the unigram update process can be eliminated, word-internal transitions, cross-word transitions to the isolated trees, and cross-word transitions to shared trees can be treated using the same process module simply. Furthermore, the internal memory usage for storing unigram transitions can be saved.

3.3 Threshold-Cutting Scheme

In the conventional architecture, the sort is processed after a Viterbi search at every frame before pruning the lower score transitions. This necessitates a large workspace because all temporal scores that are generated by the Viterbi transition must be retained until the Viterbi search of the current frame is finished at every frame, although almost all scores are pruned by the beam-cutting process at every frame. Moreover, sort processing requires computational amounts greater than 10 MIPS and consumes memory bandwidth of more than 400 MB/s for 60-k word recognition (shown in Figs. 3 and 4).

We introduce the threshold-cutting scheme to reduce the workspace and memory bandwidth instead of sort processing. In this scheme, the threshold is set adaptively to a constant value at every frame. All transitions that have a lower score than the threshold are pruned while processing Viterbi search of current frame. Only selected transitions with a higher score than the threshold are stored in workspace memory. Therefore, the proposed threshold-cutting scheme cut off the superfluous workspace.

Why is the threshold changed adaptively? This prevents degradation of the beam cutting accuracy. An improper threshold causes inconvenient cases in which too many nodes remain or too many nodes are cut off in comparison to the beam width. An adaptive threshold is set based on the difference between the average scores of the previous frame and the current frame and the number of selected transitions between the previous frame and the current frame.

Figure 5 shows beam width variation with the threshold-cut scheme when the target width is set to 1,500. The threshold cut results have ±500 variations. Regarding actual speech recognition results with a 20-k word language model, the speech recognition accuracy is unaffected by this variation of beam width because almost all transitions that engender the final speech recognition output trellis output higher scores than the others.

3.4 Two-Stage Language Model Search

The Viterbi transition comprises word-internal transitions, cross-word transitions to isolated trees and cross-word transitions to shared trees. Figure 6 presents a comparison of these transition appearance ratios derived from profiling with Julius 4.0 and Visual Studio C++. In the figure, the cross-word transitions to isolated trees are dominant. Consequently, to reduce the computational amount for the cross-word transitions to isolated trees, we propose a novel two-stage language model search scheme. This scheme is derived from the transition frequency difference between phonemic HMM and language HMM: in actual human speech, the appearance ratio of syllabic transitions is much lower than the MFCC frame rate: 100 Hz. In this scheme, the cross-word transition search is divided into two stages. The first stage is a simplified language model search for the top 10 important transitions of bigram probability. The second stage is a detailed language model search for all cross-word transitions. In the traditional language model search, only our second search treated every frame. However, in our proposed language model search, the second stage is treated at every five frames. By applying this proposed search, when the frequency of detail language search is set to 1/5, the computational amount and memory bandwidth can be reduced to 1/5, corresponding to a 78% reduction in the total Viterbi processing. The accuracy degradation that occurs when using this scheme is less than 1%, as shown by actual speech recognition measurements with the 20-k word language model.

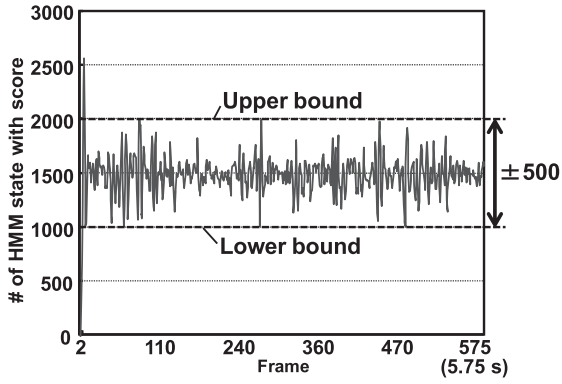


Fig. 5 Beam width variation with the threshold-cut scheme.

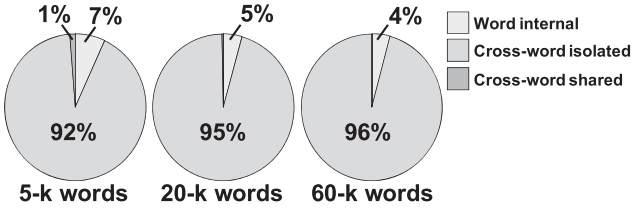


Fig. 6 Appearance ratios of three transition types in Viterbi search in Julius 4.0.

4. VLSI Architecture

The GMM memory bandwidth was reduced as explained in Sect. 3.1. Nevertheless, this reduction of GMM processing

Table 1 Memory bandwidth in 20-k word Viterbi search.

Item	Memory bandwidth [MB/s]
Tree dictionary	39.41
Transported token list	144.21
Transport information	3.09
Trellis	0.48
Bigram	346.60 (29.26)
Bigram address table	0.12
Unigram	0.35
Total	534.32(216.98)

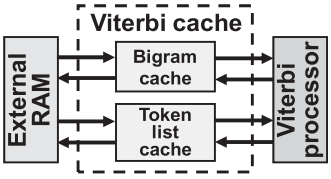


Fig. 7 Cache architecture concept in Viterbi search processing.

is not efficient when considering the total LVRCSR because the GMM memory bandwidth does not increase depending on the vocabulary library scale. However, as explained in this subsection, we specifically examine the Viterbi beam search algorithm and reduce its memory-bandwidth because the memory bandwidth of the Viterbi processor increases with the vocabulary number.

Table 1 shows the memory bandwidth of each component in Viterbi processing when considering 20-k word recognition, obtained from the hardware simulation using the referential hardware; we applied the proposed two-stage language model search to this hardware simulation. The dominant memory accesses are to the transported token lists (144.21 MB/s) that are address tables of the external memory for active state nodes, and to the bigram database (346.60 MB/s). Therefore, we introduce custom caches to these memory accesses and reduce memory bandwidth. Here, the memory bandwidth of the bigram was reduced to 29.26 MB/s, as described in the previous subsection.

Figure 7 portrays the Viterbi cache architecture concept. Two caches are introduced to the bigram and the transported token list because their memory bandwidths are wider than those for other data in Viterbi processing (shown in Table 1). The stored data are bigram probabilities for bigram cache and the temporal calculated token list for the beam cache.

Figure 8 shows a block diagram of a 60-k word speech recognition processor using our proposed schemes. To reduce the operation cycle time and external memory bandwidth, our architecture contains some schemes such as (1) cache architecture using the locality of speech recognition, (2) beam-pruning using a dynamic threshold, (3) two-stage language model search, (4) parallel GMM architecture based on mixture level and frame level, (5) parallel Viterbi architecture, and (6) pipelining operation between Viterbi

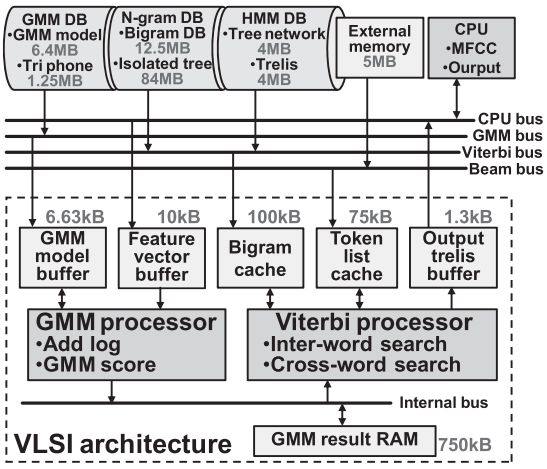


Fig. 8 Block diagram of proposed processor architecture for a 60-k word speech recognition system.

transition and GMM processing. We used an FPGA (Stratix *II*; Altera Corp.) to verify the architecture at the RTL level.

In Fig. 8, the memory sizes of three types of DBs are derived from the 2,000-state Gaussian four mixture triphone model and the 60-k word Japanese language model. The burst GMM calculation shares 50 frames, the two-stage language model search has 1/5 frequency of detail language search, and Viterbi beam width is set to 4,000. These parameters lead to the buffer sizes (6.63 kB, 10 kB and 1.3 kB), cache sizes (100 kB and 75 kB), and GMM result RAM size (750 kB), as shown in the figure. All external database sizes and the external memory size are also determined by the number of language-model vocabulary (60 k) and the Viterbi beam width (4,000).

4.1 Implementation of GMM Computation

To achieve speech recognition in real time, but at a lower operating frequency, we propose a 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 and frame-based parallel processing.
- 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 increased theoretically to the number of mixtures in the GMM. However, it increases memory bandwidth linearly. For this reason, in our architecture, the parallelism in computing the Gaussian distributions is expanded to a frame base, as described in Sect. 3.1.

As the second feature, we prepare two-input add-log units. The two-input add-log unit calculates an approximate logarithmic value of a sum of two inputs. For instance, to

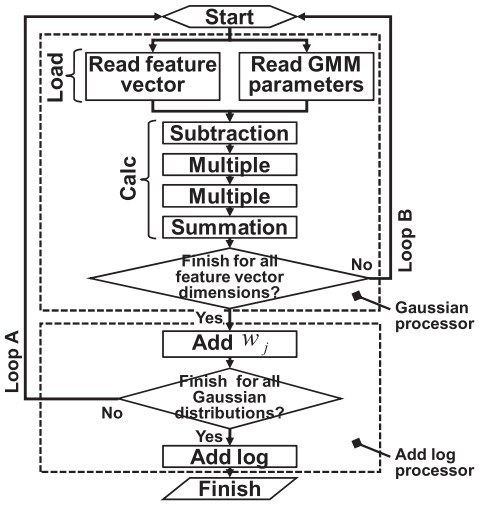


Fig. 9 GMM computation flow.

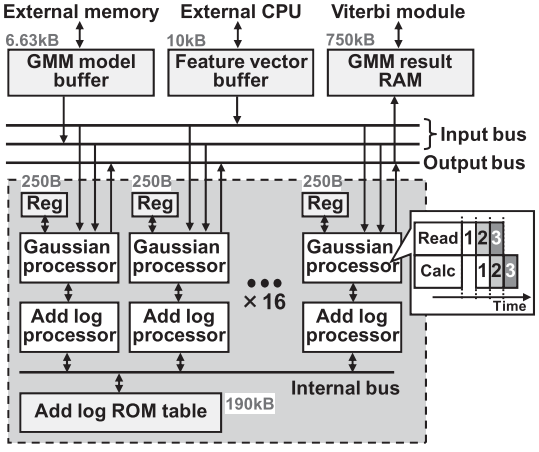


Fig. 10 GMM processor data path.

carry out four "add log"s, four data are divided into two groups: each group is input to two two-input add-log units, and each is calculated individually simultaneously. The two two-input add-log units output two results. Repeating this operation, we can obtain a desired output for any number of data. This method reduces the computation cycles. The number of parallelism in taking logarithms is 16.

The third one shows that memory reading and Gaussian distribution calculations are performed simultaneously in our dedicated hardware.

Figure 9 shows the operation flow of the GMM calculation. The GMM calculation is divisible into two steps. The first step comprises the data load and calculation processes. In our architecture, this step is treated in the Gaussian processor module. The second step is add-log computation; this step is calculated in the Add log processor module in our architecture.

Figure 10 shows the proposed GMM architecture. Input data are feature vectors among 50 frames and GMM model parameters comprising 2,000 states. Output shows the output probabilities of 50 frames that correspond to every state.

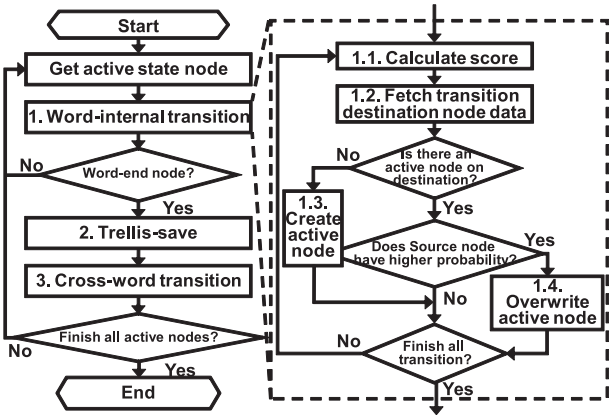


Fig. 11 Viterbi computation flow.

4.2 Viterbi and N-gram Architecture

Figure 11 presents the Viterbi transition flow. The Viterbi transition in a frame is roughly divided into three steps: word-internal transition, trellis-saving, and cross-word transition.

The Viterbi transition is performed to all active state nodes left in the previous frame. First, fetch an active state node from an active nodes queue. 1) *Word-internal* transition: perform word-internal transition when the transition source node and destination node belong in the same word. 2) *Trellis saves*: save a trellis when the active state node is the end of a word. The trellis is a dataset, with a word history and a score of the word-end node. It is used to determine the recognition result in the last frame. 3) *Cross-word transition*: perform cross-word transition after a trellis save. In this step, the transition is performed from an active state node, which is a word-end state node, to all word-beginning nodes.

The word-internal transition and cross-word transition can be expressed with the same flow. 1.1) *Calculate score*: The transition probability from the HMM dictionary is added to the score of active state node. 1.2) *Fetch transition destination node date*: fetch the information of a transition destination node from a HMM dictionary and Result RAM. 1.3) *Create active state node*: create an active state node when no active state node exists on the transition destination. 1.4) *Overwrite active state node*: overwrite the active state node on destination when a lower probable active state node exists on the transition destination.

Figure 12 portrays the proposed architecture for reducing the Viterbi processor memory bandwidth. The proposed architecture employs a specialized cache, threshold cut, and two-stage language model search. The Viterbi cache is set between the external memory and Viterbi processor.

The following subsections show the specialized cache architecture in detail. The number of words in the vocabulary dictionary is set to 20,000. The beam width is set to 2,000. The start nodes are 1,000. To limit the memory

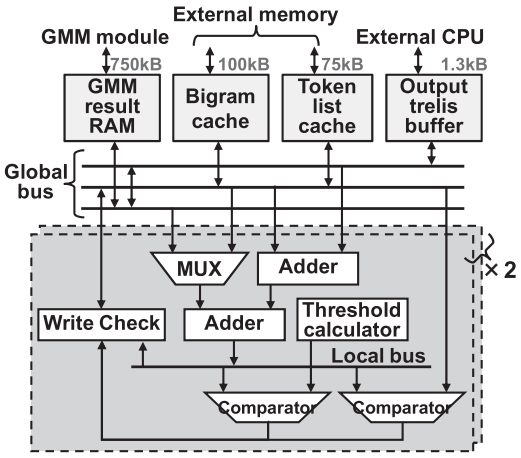


Fig. 12 Proposed Viterbi architecture.

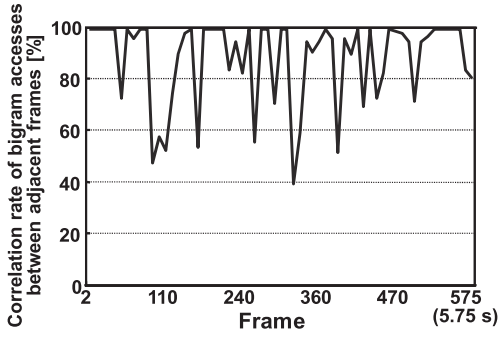


Fig. 13 Correlation rate of bigram accesses between adjacent frames profiled with Julius 4.0.

capacity of buffered RAM for VLSI, a Viterbi processor is necessary for efficient processing by implementing a specialized cache.

4.3 Bigram Cache

Locality of data is important to implement a cache. However, no chance of reading the same bigram probability exists in a frame in HMM algorithm. Therefore, to implement a bigram cache, data correlation between frames is important. Figure 13 shows the correlation that we researched by using software profiling with Visual C++ and Julius 4.0. The figure means that a correlation exists between frames, which accounts for about 60–90%. From this research, we decided that the target value of the hit rate is about 70%.

The top 10 data are read in using a two-stage language model search process. We then decide that the cache line size is 40 B (= 10 × 4 B) to fit it. Figure 14 shows the relation between the hit rate and cache size with a direct mapping scheme by hardware-emulated profiling using Visual C++ based on custom array corresponding to cache architecture. Additionally, to achieve a higher hit rate, we employ another scheme using a feature of speech recognition flow. In the bigram step, a node with a higher score tends to survive in the next frame. To set the timing of writing a score

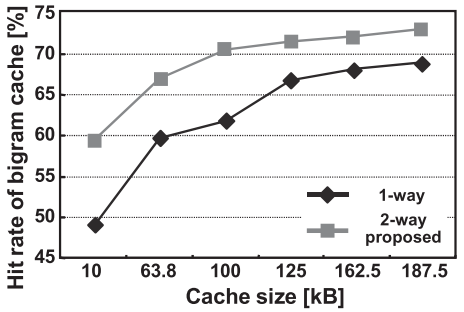


Fig. 14 Bigram cache hit rate profiled with Julius 4.0.

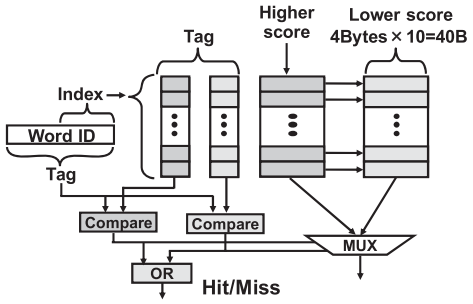


Fig. 15 Proposed two-way cache.

to the cache as the score is overwritten, the higher score is stored in the cache late in the same frame because the existing score is overwritten when a higher score appears. Using such features of speech recognition, we propose a cache as shown below.

The proposed cache scheme is presented in Fig. 15. The cache adopts a two-way set associative scheme. In the two-way set associative scheme in Fig. 15, a lower bit of word ID is used to create an index, and the whole word ID is used as a tag. The two-way consists of two parts: one for a higher score and one for a lower score. When a mis-hit occurs, a new bigram probability is stored in the high score part because the bigram probability of late in the frame is higher. On this occasion, the score in the higher part is settled to the lower part; a new bigram score is stored in the higher part. In doing so, a bigram probability computed late in the frame tends to remain in the cache. Therefore, the hit rate becomes high. Figure 14 shows the hit rate. Using the proposed architecture in this paper, we decide that the cache size is 100 kB to achieve the target hit rate of 70%. By implementing the bigram cache and two-stage language model search, we can reduce the memory bandwidth of bigram probability by about 94%.

4.4 Token List Cache

We adopt a direct mapping cache for the token list cache. The cache line size is set to 4 bits, which is the same as the size of a transitioned token list data. In our architecture, to guarantee high-accuracy recognition, the beam width is targeted to 2,000. In fact, 60% of node information is occupied by start-node data, and the bigram calculates frequently

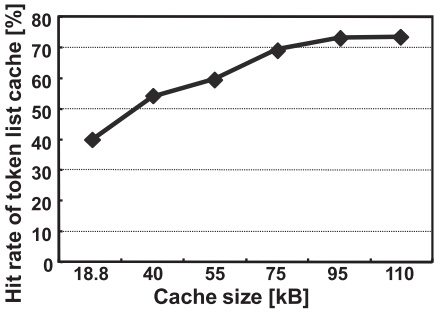


Fig. 16 Token list cache hit rate.

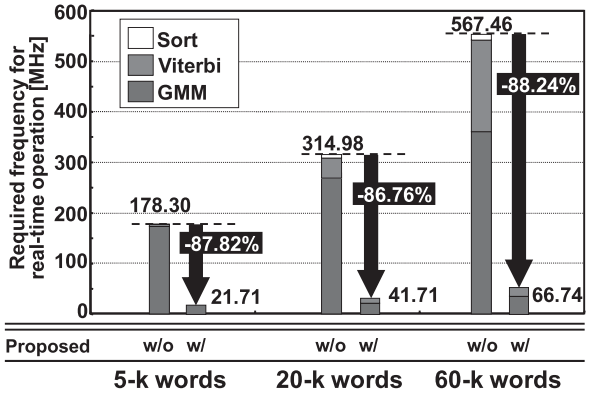


Fig. 17 Required frequency comparison with conventional architecture.

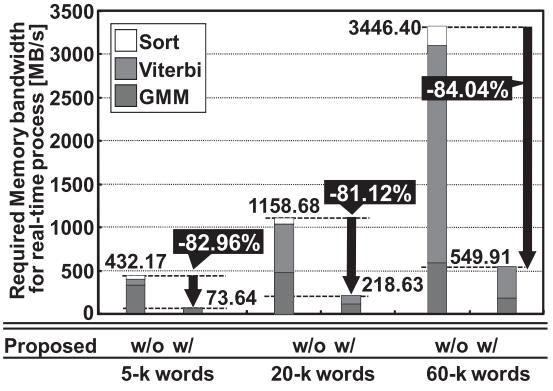


Fig. 18 Required memory bandwidth using the real-time process.

accessed start-node data. Therefore, the start-node data always hold in the token list cache. The hit rate of the token list cache is presented in Fig. 16. Using the architecture proposed in this paper, we decide that the cache size is 75 kB to achieve the target hit rate of 70%.

5. Implementation Results

As described in Sect. 2, we first implemented this architecture on an FPGA (Stratix *II*; Altera Corp.) to verify the VLSI architecture at the register transfer level (RTL) basis. However, our VLSI architecture described in Sect. 4 has to utilize a larger size of internal memory than the FPGA has. Therefore, to exploit the cache operation and its effect, we simulate SRAM models and corresponding logic operation in speech recognition using a Verilog simulator, and obtain a required frequency and memory bandwidth for real-time operation.

5.1 Required Frequency and Memory Bandwidth

Figure 17 shows the required operating frequencies of the proposed architecture and referential architecture (shown in Fig. 3) in each number of vocabularies. The proposed architecture incorporates the proposed schemes described in Sect. 3 (the referential hardware does not). Especially, the two-stage language model search in the proposed architecture contributes to reduce the required frequency by almost "1/5". In Fig. 17, our proposed architecture can operate real-time speech recognition at 21.71 MHz for 5-k words, at 41.71 MHz for 20-k words, and at 66.74 MHz for 60-k words. Our proposed architecture can reduce the required frequency for real-time speech recognition by 87.82% for 5-k words, by 86.76% for 20-k words, and by 88.24% for 60-k words.

Figure 18 shows the required memory bandwidth of our proposed architecture and referential architecture and referential architecture (shown in Fig. 4). Our proposed architecture achieves 82.96% reduction (73.64 MB/s) for 5-k words, 81.12% reduction (218.63 MB/s) for 20-k words, and 84.04% reduction (549.91 MB/s) for 60-k words.

5.2 Comparison with Other Architectures

Table 2 presents a comparison of specifications with other hardware-based systems. This table shows the vocabulary size, GMM model, Viterbi beam width, accuracy, real-time factor frequency, memory bandwidth, logic size, memory size, and platform. The vocabulary size represents the number of words in the language model dictionary. The accuracy is the recognition rate. The real-time factor means how fast the hardware is: for example, a real-time factor of "0.5" corresponds to "×2" faster than a real-time operation. The frequency represents the operating frequency of the hardware speech recognition system. The external memory bandwidth equals the data transfer traffic between an FPGA/VLSI chip and external SDRAM/DRAM memory.

The comparison reveals that our architecture achieves the lowest external memory bandwidth with the same-size vocabulary. Our architecture reduces the required minimum operating frequency and external memory bandwidth for real-time operation to 58% (= 41.71/72) and 38% (= 218.63/576) in the 20-k word speech recognition, respectively, compared to the SNU architecture (required frequency: 72 MHz, and external memory bandwidth: 576 MB/s, considering the real-time factor: 0.72) [7]. This is because, our architecture utilizes the advantages derived from the novel techniques: the burst GMM calculation (described in Sect. 3.1), the two-stage language model search (Sect. 3.4), and the specific cache implementation (denoted in Sects. 4.3 and 4.4). These techniques enable

Table 2 Comparison with other hardware-based systems.

	This work			Nagoya [9]	Intel (GMM only) [8]	SNU [7]	CMU [4]	CMU(Viterbi only)[5]	
								Single FPGA	Dual FPGAs
Vocabulary (k)	5	20	60	0.8	20	20	1	5	5
GMM model # of states	2,000			32	8,000	3,001	NA	4,147	
GMM model # of distributions	16			NA	16	16	8	NA	
GMM model # of dimensions	25			38	39	39	39	NA	
Viterbi beam width	500	2,000	4,000	NA	NA	500	NA	NA	
Accuracy (%)	80.13	87.81	89.99	NA	90.10	87.31	89.10	92.28	92.28
Real-time factor	1	1	1	1	1.15	0.72	2.20	0.17	0.10
Frequency (MHz)	21.71	41.71	66.74	2.69	1600	100	50	100	100
External memory BW (MB/s)	73.64	218.63	549.91	NA	480	800	100	558	950
Resources	75,490 LUTs			NA	NA	13,835 slices	13,449 slices	25,973 slices	2,812,480 slices
Internal memory (kB)	458	778	1,133	1.94	NA	52	305	566	993
External memory (MB)	11.00	27.62	103.78	NA	NA	49.44	3	88	176
Platform	Altera Stratix *II*			NA	Atom Z530	Xilinx Virtex-4	Xilinx XUP	Xilinx Virtex *II* Pro70	

our architecture to process 60-k word speech recognition in real time. Our architecture can be easily applied to limited hardware-resource devices, such as ubiquitous/wearable applications, because of its low external memory bandwidth with a large vocabulary language model.

6. Conclusion

We proposed a VLSI architecture to support real-time continuous speech recognition. To reduce the operation cycle time and external memory bandwidth, our architecture contains cache architecture using the locality of speech recognition, beam pruning using a dynamic threshold, two-stage language model search, parallel GMM architecture based on mixture level and frame level, parallel Viterbi architecture, and pipeline operation between Viterbi transition and GMM processing. Results show that our architecture achieves 88.24% required frequency reduction (66.74 MHz) and 84.04% memory bandwidth reduction (549.91 MB/s) for real-time 60-k word continuous speech recognition.

Acknowledgments

This work is supported by VLSI Design and Education Center (VDEC), the University of Tokyo in collaboration with Mentor Graphics, Inc. and Synopsys, Inc.

This work was supported by KAKENHI (18200003).

The authors appreciate valuable discussions with Prof. Yasuo Ariki related to speech recognition algorithm.

References

[1] K. Miura, H. Noguchi, H. Kawaguchi, and M. Yoshimoto, "A low memory bandwidth Gaussian mixture model (GMM) processor for 20,000-word real-time speech recognition FPGA system," Proc. IEEE Intl. Conf. on Field-Programmable Technology (FPT), pp.341–344, Dec. 2008.

[2] T. Fujinaga, K. Miura, H. Noguchi, H. Kawaguchi, and M. Yoshimoto, "Parallelized Viterbi processor for 5,000-word large-vocabulary real-time continuous speech recognition FPGA system," Proc. ISCA Annual Conf. of Intl. Speech Communication Association (Interspeech), pp.1483–1486, Sept. 2009.

[3] A. Lee, T. Kawahara, and K. Shikano, "Julius — An open source real-time large vocabulary recognition engine," Proc. European Conf. on Speech Communication and Technology (EUROSPEECH), pp.1691–1694, Sept. 2001.

[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," Proc. 2007 ACM/SIGDA 15th Intl. Symposium on Field Programmable Gate Arrays (FPGA), pp.60–68, Feb. 2007.

[5] E.C. Lin and R.A. Rutenbar, "A multi-FPGA 10x-real-time high-speed search engine for a 5000-word vocabulary speech recognizer," Proc. ACM/SIGDA Intl. Symposium on Field Programmable Gate Arrays (FPGA), pp.83–92, Feb. 2009.

[6] Y. Choi, K. You, and W. Sung, "FPGA-based implementation of a real-time 5000-word continuous speech recognizer," Proc. 16th European Signal Processing Conf. (EUSIPCO), Aug. 2008.

[7] Y. Choi, K. You, J. Choi, and W. Sung, "A real-time FPGA-based 20,000-word speech recognizer with optimized DRAM access," IEEE Trans. Circuits Syst. I, vol.57, no.8, pp.2119–2131, Feb. 2010.

[8] T. Ma and M. Deisher, "Novel CI-backoff scheme for real-time embedded speech recognition," Proc. IEEE Intl. Conf. on Acoustics, Speech, and Signal Processing (ICASSP), pp.1614–1617, March 2010.

[9] K. Nakamura, M. Yamamoto, K. Takagi, and N. Takagi, "A VLSI architecture for outout probability computations of HMM-based recognition systems with store-based block parallel processing," IEICE Trans. Inf. & Syst., vol.E93-D, no.2, pp.300–305, Feb. 2010.



Hiroki Noguchi received his B.E. and M.E. degrees in Computer and Systems Engineering in 2006 and 2008, respectively from Kobe University, Hyogo, Japan, where he is currently earning a Ph.D. degree. His research interests are low-power SRAM designs, multimedia/ubiquitous systems and digital signal processing architectures, which include speech-recognition for handheld, image-recognition for wearable computing, and mixed integer programming for real-time robotics controlling, and their low-power hardware implementation. He is a student member of IEEE.



Kazuo Miura received his B.E. and M.E. degrees in Computer and Systems Engineering, respectively from Kobe University, Hyogo, Japan, in 2008 and 2010. He is currently working at Simplex Technology Corporation. His research interests are ultra-low-power techniques in digital LSIs, and speech recognition algorithms and its hardware implementation.



Tsuyoshi Fujinaga received his B.E. degree in Computer and Systems Engineering from Kobe University, Hyogo, Japan, in 2009. He is currently working in the M.E. course at that university. His current research is speech recognition algorithms and its hardware implementation for wearable devices.



Takanobu Sugahara was born on Apr. 24, 1987. He received his B.E. degree in Computer and Systems Engineering in 2010 from Kobe University, Hyogo, Japan, where he is currently working in the M.E. course. His current research is speech recognition algorithms and its hardware implementation for wearable devices.



Hiroshi Kawaguchi received his B.E. and M.E. degrees in Electronic Engineering from Chiba University, Chiba, Japan, in 1991 and 1993, respectively, and earned a Ph.D. degree in Engineering from The University of Tokyo, Tokyo, Japan, in 2006. He joined Konami Corporation, Kobe, Japan, in 1993, where he developed arcade entertainment systems. He moved to the Institute of Industrial Science, The University of Tokyo, as a Technical Associate in 1996, and was appointed as a Research Associate in 2003. In 2005, he moved to Kobe University, Kobe, Japan. Since 2007, he has been an Associate Professor with the Department of Information Science at that university. He is also a Collaborative Researcher with the Institute of Industrial Science, The University of Tokyo. His current research interests include low-voltage SRAM, RF circuits, and ubiquitous sensor networks. Dr. Kawaguchi was a recipient of the IEEE ISSCC 2004 Takuo Sugano Outstanding Paper Award and the IEEE Kansai Section 2006 Gold Award. He has served as a Design and Implementation of Signal Processing Systems (DISPS) Technical Committee Member for IEEE Signal Processing Society, as a Program Committee Member for IEEE Custom Integrated Circuits Conference (CICC) and IEEE Symposium on Low-Power and High-Speed Chips (COOL Chips), and as a Guest Associate Editor of IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences and IPSJ Transactions on System LSI Design Methodology (TSLDM). He is a member of the IEEE, ACM and IPSJ.



Masahiko Yoshimoto earned his B.S. degree in Electronic Engineering from Nagoya Institute of Technology, Nagoya, Japan, in 1975, and the M.S. degree in Electronic Engineering from Nagoya University, Nagoya, Japan, in 1977. He earned a Ph.D. degree in Electrical Engineering from Nagoya University, Nagoya, Japan in 1998. He joined the LSI Laboratory, Mitsubishi Electric Corporation, Itami, Japan, in April 1977. During 1978–1983 he was engaged in the design of NMOS and CMOS static RAM including a 64K full CMOS RAM with the world's first divided-wordline structure. From 1984, he was involved in research and development of multimedia ULSI systems for digital broadcasting and digital communication systems based on MPEG2 and MPEG4 Codec LSI core technology. Since 2000, he has been a Professor of the Department of Electrical and Electronic Systems Engineering at Kanazawa University, Japan. Since 2004, he has been a Professor of the Department of Computer and Systems Engineering at Kobe University, Japan. His current activities are focused on research and development of multimedia and ubiquitous media VLSI systems including an ultra-low-power image compression processor and a low-power wireless interface circuit. He holds 70 registered patents. He served on the Program Committee of the IEEE International Solid State Circuit Conference from 1991 to 1993. Additionally, he served as a Guest Editor for special issues on Low-Power System LSI, IP, and Related Technologies of IEICE Transactions in 2004. He received R&D100 awards in 1990 and 1996 from R&D Magazine for development of the DISP and development of a real-time MPEG2 video encoder chipset, respectively.