Vol.2010-SLDM-144 No.8
Vol.2010-EMB-16 No.8
Vol.2010-MBL-53 No.8
Vol.2010-UBI-25 No.8
2010/3/26

保存型一括並列処理による高速なHMM出力確率計算・最尤推定回路の構成法

島崎 亮[†1] 中村一博[†1] 山本正俊[†1]
高木一義[†1] 高木直史[†1]

本稿では、保存型一括並列処理に適した最尤推定のハードウェアアルゴリズムと、保存型一括並列処理における HMM(隠れマルコフモデル) 出力確率計算の高速化法、それらに基づく高速な HMM 出力確率計算・最尤推定回路の VLSI アーキテクチャを提案する。提案する最尤推定のハードウェアアルゴリズムにより、保存型一括並列処理による HMM 出力確率計算と、その結果を用いる最尤推定のパイプライン処理が可能になる。提案する HMM 出力確率計算の高速化手法により、従来の保存型一括並列処理では導入しても並列に動作させることができなかった PE(Processing Element) の並列動作が可能になり、より多くの PE を動かすことによる HMM を用いた認識処理の高速化が期待できる。

A Fast VLSI Architecture of Output Probability Computations and Viterbi Scorer for HMM-Based Recognition Systems with Store-Based Block Parallel Processing

Ryo Shimazaki, Kazuhiro Nakamura,
Masatoshi Yamamoto, Kazuyoshi Takagi
and Naofumi Takagi

In this paper, We present a fast VLSI architecture for output probability computations of continuous Hidden Markov Models (HMMs) and Viterbi scorer with *store-based block parallel processing* (*StoreBPP*). We also demonstrate *fast store-based block parallel processing* (*FastStoreBPP*) which exploits full performance of the StoreBPP.

1. Introduction

Due to their effectiveness and efficiency for user-independent recognition, hidden Markov models (HMMs) are widely used in applications such as speech recognition (word recognition, connected word recognition, continuous speech recognition), lip-reading, and gesture recognition. Output probability computations and Viterbi scorer are the most time-consuming part of HMM-based recognition systems.

High-speed VLSI architectures optimized for recognition tasks have been developed[1)–7)] for the development of well-optimized HMM-based recognition systems. Mathew *et al*. developed accelerators for the SPHINX 3[8)] speech recognition system[6)] and perception accelerators for embedded systems[7)]. Yoshizawa *et al*. investigated a *block-wise parallel processing* for output probability computations of continuous HMMs and Viterbi scorer, and proposed a high-speed VLSI architecture[1)–3)]. Nakamura *et al*. also investigated a block-wise parallel processing method, *store-based block parallel processing* (*StoreBPP*), for output probability computations of continuous HMMs, and proposed a high-speed VLSI architecture without Viterbi scorer[5)]. Different block parallel processing methods require different architectures of Viterbi scorer. Viterbi scorer which is suitable for the StoreBPP architecture is required for the development of well-optimized future HMM-based recognition systems.

In this paper, a pipelined VLSI architecture of Viterbi scorer for StoreBPP is presented. We also demonstrate *fast store-based block parallel processing* (*FastStoreBPP*) for output probability computations of HMMs and Viterbi scorer, and present a VLSI architecture that supports it, which exploits full performance of the StoreBPP.

Compared with the conventional StoreBPP[5)] and *StreamBPP*[1)], the proposed architecture requires fewer registers and processing elements and less processing time. A comparison demonstrates the efficiency of the proposed architecture. The results show that full performance of the StoreBPP has been exploited by the FastStoreBPP architecture which extends the bit length of the input bus (e.g. 8-bit to 16-bit).

†1 名古屋大学
Nagoya University

Vol.2010-SLDM-144 No.8
Vol.2010-EMB-16 No.8
Vol.2010-MBL-53 No.8
Vol.2010-UBI-25 No.8
2010/3/26

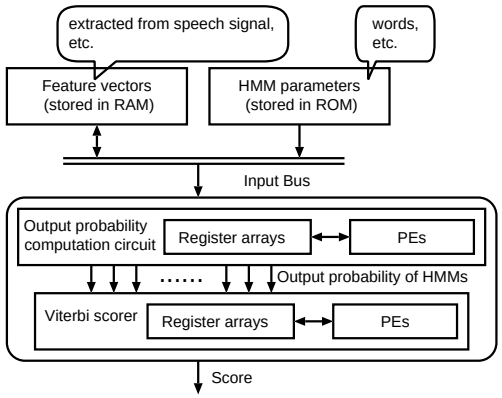

Fig. 1 Basic structure of HMM-based recognition hardware.

2. HMM-based Recognition Systems

2.1 HMM-based Recognition Hardware

Figure 1 shows the basic structure of HMM-based recognition hardware[1)–7)]. The output probability computation circuit and Viterbi scorer work together as a recognition engine. The inputs to the output probability computation circuit are feature vectors of several dimensions and model parameters of HMMs. These values are stored in RAM and ROM respectively. The RAM, ROM, output probability computation circuit and Viterbi scorer interconnect via a single bus, and memory accesses are exclusive. The output probability computation circuit outputs the results of the output probability computation of HMMs. The Viterbi scorer outputs likelihood score using the Viterbi algorithm. In HMM-based recognition systems, the most time-consuming task is output probability computations and likelihood score computations, and the output probability computation circuit and the Viterbi scorer accelerate these computations. The output probability computation circuit and the Viterbi scorer have several register arrays and processing elements (PEs) for efficient high-speed parallel processing.

2.2 Output Probability Computation of HMMs and Likelihood Score Computation with Viterbi Algorithm

Let $\mathbf{O}_1,\ \mathbf{O}_2,\ ...,$ and $\mathbf{O}_T$ be a sequence of P-dimensional input feature vectors to HMMs, where $\mathbf{O}_t = (o_{t1},\ o_{t2},\ ...,\ o_{tP}),\ 1 \le t \le T$. T is the number of input feature vectors, and P is the dimension of the input feature vector. For an $\mathbf{O}_t$, the output probability of N-state left-to-right continuous HMM at the j-th state is given by

$$\log b_j(\mathbf{O}_t) = \omega_j + \sum_{p=1}^{P} \sigma_{jp}(o_{tp} - \mu_{jp})^2, \quad 1 \le j \le N,\ 1 \le t \le T, \tag{1}$$

where ω_j, σ_{jp}, and μ_{jp} are the factors of the Gaussian probability density function.

The output probability computation circuit (Fig. 1) computes $\log b_j(\mathbf{O}_t)$ based on Eq. (1), where all HMM parameters ω_j, σ_{jp}, and μ_{jp} are stored in ROM, and the input feature vectors are stored in RAM. The values of T, N, P, and the number of HMMs V differ for each recognition system. For a recent isolated word recognition system[1),2)], T, N, P, and V are 86, 32, 38, and 800, respectively, and for another word recognition system[3)], T, N, P, and V are 89, 12, 16 and 100.

For output probabilities $\log b_j(\mathbf{O}_t),\ 1 \le j \le N, 1 \le t \le T$, log-likelihood score $\log P^*$ is is given by

$$\log \delta_1(j) = \log \pi_j + \log b_j(\mathbf{O}_1) \tag{2}$$

$$\log \delta_t(j) = \min[\log \delta_{t-1}(j-1) + \log a_{j-1,j}, \log \delta_{t-1}(j) + \log a_{j,j}] + \log b_j(\mathbf{O}_t) \tag{3}$$

$$\log P^* = \min_{1 \le j \le N} [\log \delta_T(j)] \tag{4}$$

using Viterbi algorithm in HMM-based recognition hardware.[1)–4)]. A flowchart of output probability computation and likelifood score computations is also shown in Fig. 2. Likelifood scores are obtained by $N \cdot T \cdot V$ times the partial computation of $\log \delta_t(j)$ calls. Partial computation of $\log b_j(\mathbf{O}_t)$ performs 4 arithmetic operations, an addition, a subtraction and two multiplications for Eq. (1) and computes $\log b_j(\mathbf{O}_t)$. Partial computation of $\log \delta_t(j)$ performs 3 arithmetic operations, three additions for Eq. (2), (3), (4), and computes $\log \delta_T(j)$.

3. Fast VLSI Architecture of Output Probability Computations and Viterbi Scorer with Fast Store-Based Block Parallel Processing

3.1 VLSI Architecture of Viterbi Scorer for StoreBPP

Block parallel processing (BPP) for output probability computations and Viterbi scorer was proposed as an efficient parallel processing method for word HMM-based speech recognition by Yoshizawa *et al.*[1)–3)]. In this method, the set of input feature

Vol.2010-SLDM-144 No.8
Vol.2010-EMB-16 No.8
Vol.2010-MBL-53 No.8
Vol.2010-UBI-25 No.8
2010/3/26

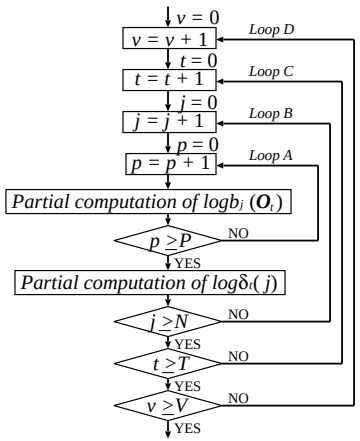

Fig. 2 Flowchart of output probability computation and likelihood score computation.

vectors is called a *block*, and HMM parameters are effectively shared between different input feature vectors in the computation. N-parallel computation is performed by their BPP, and in recent years, two types of BPP are classified according to input data flow: stream-based block parallel processing (StreamBPP) and store-based block parallel processing (StoreBPP) by Nakamura *et al.*[5]. A block can be seen as a set of $M(\leq T)$ input feature vectors, whose elements are $\mathbf{O}_{t'}$'s, $1 \leq t' \leq M$. M vectors in T input feature vectors are processed in block. StoreBPP performs arithmetic operations to locally stored input feature vectors, which are $\mathbf{O}_1$, $\mathbf{O}_2$, ..., and $\mathbf{O}_M$. On the other hand, a block can also be seen as a $M \times P$ matrix whose elements are $o_{t'p}$, $1 \leq t' \leq M$, $1 \leq p \leq P$. StreamBPP performs arithmetic operations to an input stream, which is o_{11}, ..., o_{1P}, o_{21} ..., o_{2P}, ..., o_{M1} ..., o_{MP}. The BPP proposed by Yoshizawa *et al.*[1)–3)] is classified as a StreamBPP for output probability computations and Viterbi scorer. The BPP proposed by by Nakamura *et al.*[5)] is classified as a StoreBPP for output probability computations. $M/2$-parallel computations are performed by the StoreBPP.

A flowchart of the output probability computations and Viterbi scorer with the StreamBPP[1)–3)] is shown in Fig. 3. $PE1j$ represents the j-th processing element, which computes $\log b_j(\mathbf{O}_t)$ based on Eq. (1). $PE2j$ represents the j-th processing element,

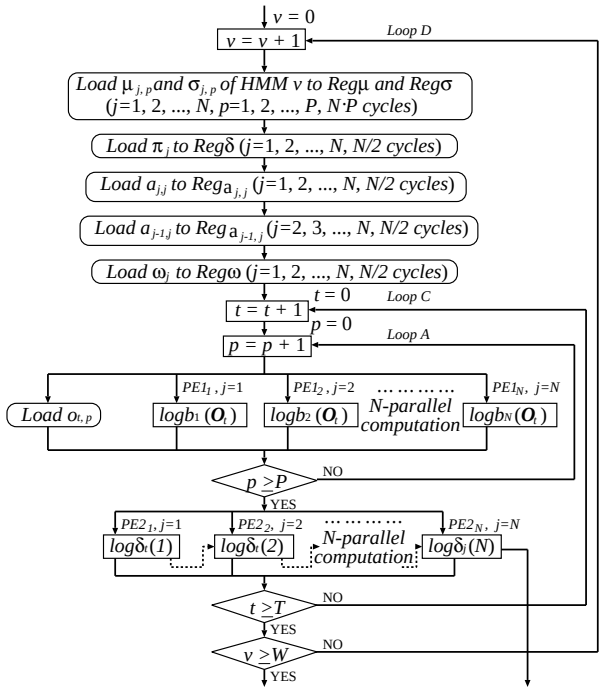

Fig. 3 Flowchart of output probability computation and Viterbi scorer using StreamBPP.

which computes $\log P^*$ by three additions for Eq. (2), (3) and (4). Loop B (Fig. 2) is expanded as shown in Fig. 3, and $\log b_1(\mathbf{O}_t)$, $\log b_2(\mathbf{O}_t)$, ..., and $\log b_N(\mathbf{O}_t)$ are computed simultaneously with N $PE1s$. In addition to the N-state parallel computation, the same HMM parameters μ_{jp}'s, σ_{jp}'s, and ω_j's, $1 \leq j \leq N$, $1 \leq p \leq P$, are used repeatedly during Loop C in Fig. 3.

A flowchart of the output probability computation with StoreBPP[5)] is shown in Fig. 4. The $PE1s$ and $PE2s$ in Figs. 4 and 3 are identical. Loop C in Fig. 2 is partially expanded in Fig. 4, and $\log b_j(\mathbf{O}_{t'+1})$, $\log b_j(\mathbf{O}_{t'+2})$, ..., and $\log b_j(\mathbf{O}_{t'+M/2})$ are computed simultaneously with $M/2$ $PE1s$ in Loop C1. In addition to the $M/2$-parallel computations, $\log b_j(\mathbf{O}_{t'+M/2+1})$, ..., and $\log b_j(\mathbf{O}_{t'+M})$ are also computed with the same $M/2$ $PE1s$. In this double $M/2$-parallel computation, the same HMM parameters μ_{jp} and σ_{jp} are used twice, because the parameters are independent of t. In addition to the

Vol.2010-SLDM-144 No.8
Vol.2010-EMB-16 No.8
Vol.2010-MBL-53 No.8
Vol.2010-UBI-25 No.8
2010/3/26

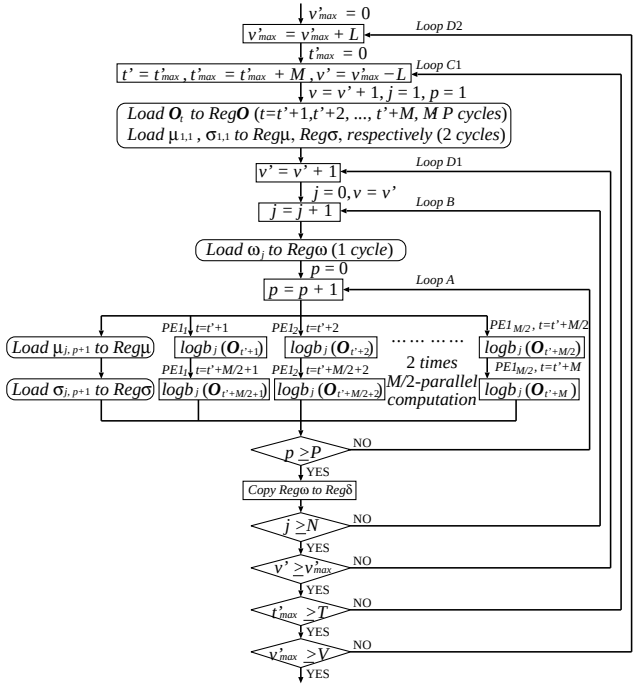

Fig. 4 Flowchart of output probability computations using StoreBPP.

$M/2$-parallel computations, Loop D (Fig. 2) is divided into Loops D1 and D2 (Fig. 4). The same feature vectors $\mathbf{O}_{t'+1}$, ..., and $\mathbf{O}_{t'+M}$ are used repeatedly during Loop D1.

For j-th HMM state, M output probabilities $\log b_j(\mathbf{O}_t)$, $t = t'+1, \cdots, t'+M$, are simultaneously obtained by Loop A. Different block parallel processing methods require different architectures of Viterbi scorer. Viterbi scorer which is suitable for the StoreBPP architecture is required for the development of well-optimized future HMM-based recognition systems.

We present a pipelined VLSI architecture of Viterbi scorer for the StoreBPP. A flowchart of the output probability computation and Viterbi scorer with the StoreBPP is shown in Fig. 5. The $PE1s$ and $PE2s$ in Figs. 5, 4 and 3 are identical. Loop A in Figs. 4 and Figs. 3 are also identical. Likelihood scores are computed by $\lceil M/P \rceil$ Loop A' with pipelined $\lceil M/P \rceil$ $PE2s$ based on Eq. (2), (3) and (4). For $(j-1)$-th

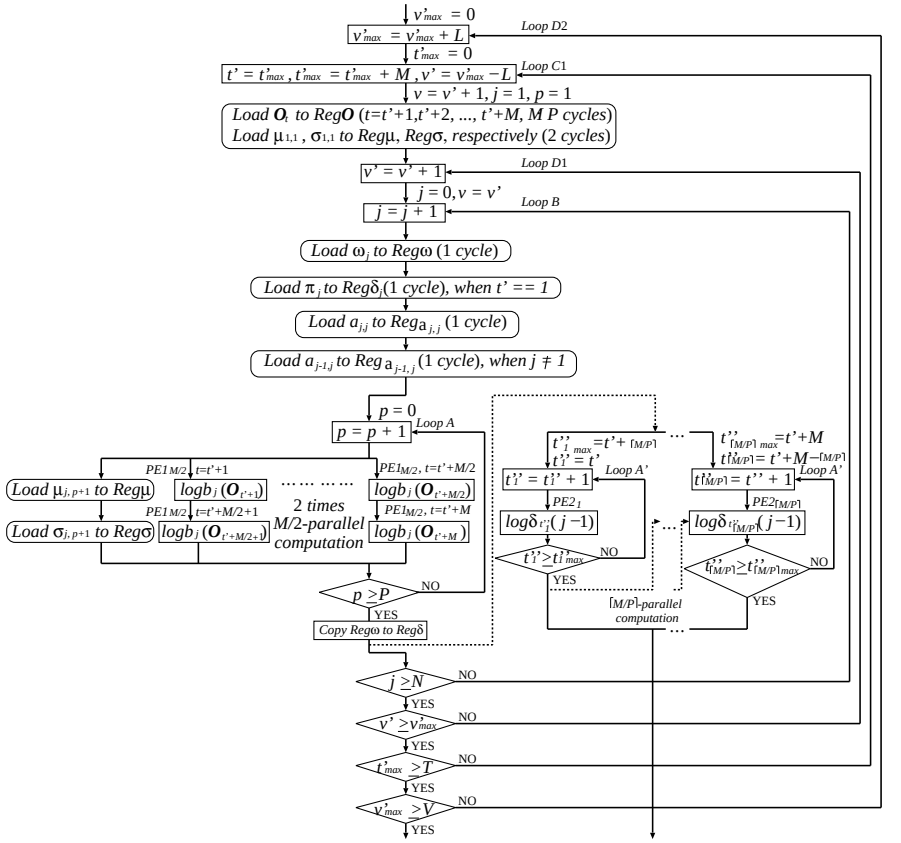

Fig. 5 Flowchart of output probability computations and Viterbi scorer using StoreBPP.

HMM state, $\log \delta_{t''}(j-1)$ is computed with $\log \delta_{t''-1}(j-2)$ and $\log \delta_{t'-1}(j-1)$ during in Loop A'. The intermediate result score $\log \delta_{t'}(j-1)$ has to be computed during $M/2$-parallel computation of Loop A.

3.2 Fast Store-based Block Parallel Processing (FastStoreBPP) and a VLSI Architecture that Supports It

We demonstrate *fast store-based block parallel processing* (*FastStoreBPP*) for output

Vol.2010-SLDM-144 No.8
Vol.2010-EMB-16 No.8
Vol.2010-MBL-53 No.8
Vol.2010-UBI-25 No.8
2010/3/26

probability computations of HMMs and Viterbi scorer, and present a VLSI architecture that supports it. A flowchart of the output probability computation and Viterbi scorer with FastStoreBPP is shown in Fig. 6. Output probabilities are computed by Loop A with M $PE1s$. Likelihood scores are computed by $\lceil M/P \rceil$ Loop A' with pipelined $\lceil M/P \rceil$ $PE2s$ based on Eq. (2), (3) and (4). In the StoreBPP(Fig. 5), the M calculations of $\log b_j(\mathbf{O}_t)$, $t' + 1 \leq t \leq t' + M$, were performed with 2 cycles, because it requires 1 cycle to read parameters which will be needed for the next calculation. In Loop A(Fig. 5), the M calculations ware proceeded with $M/2$ $PE1s$, where $M/2$ was adequate number for the calcuations. The objective of the FastStoreBPP is efficient high-speed parallel processing of StoreBPP by a little extention of the bit length of the input bus(Fig. 2). By extending the input bus, it can read two parameters at once, therefore M calculations of $\log b_j(\mathbf{O}_t)$, $t' + 1 \leq t \leq t' + M$, were performed with 1 cycles, and the M calculations are proceeded with M $PE1s$. where M is adequate number for the calcuations.

Our FastStoreBPP VLSI architecture for output probability computations and Viterbi scorer is shown in Fig. 7, where the number of $PE1s$ $M \leq P$. The architecture has two register arrays, two registers and M $PE1s$ for output probability computations. The architecture has four register arrays, two registers and one $PE2s$ for likelihood score computations based on Eq. (2), (3) and (4). Reg$\mathbf{O}$ stores M input feature vectors $\mathbf{O}_{t'+1}$, $\mathbf{O}_{t'+2}$, ..., and $\mathbf{O}_{t'+M}$. Regμ and Regσ store HMM parameters $-\mu_{jp}$, and σ_{jp}, respectively. Regω stores HMM parameter ω_j and intermediate results. Regδ stores computed output probabilities for a Viterbi scorer. Regδ_l stores intermediate results $\log \delta_{t'+M}(j)$, $1 \leq j \leq N$, of L HMMs. Regδ_j and Regδ_{j-1} store intermediate results $\log \delta_t(j)$ and $\log \delta_t(j-1)$, $t' + 1 \leq t \leq t' + M$, of v-th HMM. Rega$\mathbf{a}_{j,j}$ and Rega$\mathbf{a}_{j-1,j}$ store HMM parameters $\log a_{j,j}$ and $\log a_{j-1,j}$ of an HMM, respectively. Each $PE1$ consists of two adders and two multipliers, which are used for computing Eq. (1). The computation starts by reading M input feature vectors from RAM and storing them to Reg$\mathbf{O}$ in Loop C1 (Fig. 6). The HMM parameters of v-th HMM are read from ROM and stored in Regμ, Regσ, Regω, Regδ_j and Rega$\mathbf{a}_{j,j}$, which are μ_{11}, σ_{11}, ω_1, π_1 and a_{11}. For the M stored input feature vectors $\mathbf{O}_{t'+1}$, $\mathbf{O}_{t'+2}$, ..., and $\mathbf{O}_{t'+M}$, M intermediate results are simultaneously computed with the stored μ_{11} and σ_{11} by

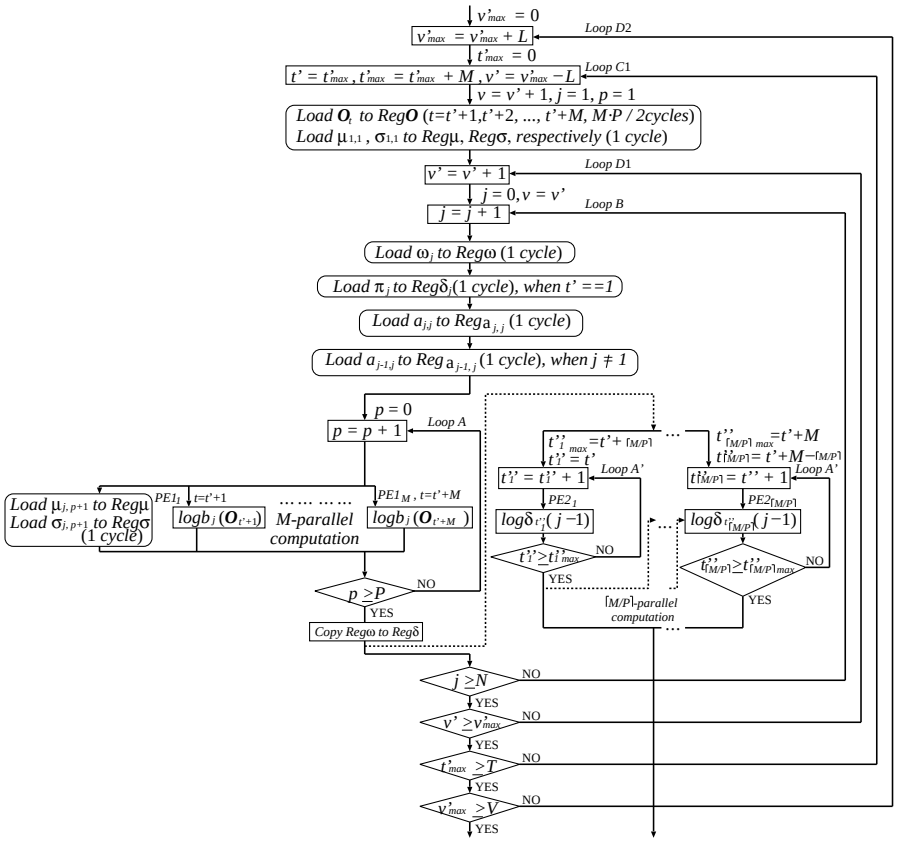

Fig. 6 Flowchart of output probability computations and Viterbi scorer using FastStoreBPP architecture.

M $PE1s$, where the HMM parameters are shared by all $PE1s$. At the same time, an HMM parameter μ_{jp+1} and σ_{jp+1} of v-th HMM are read from ROM and stored in Regμ and Regσ. In this M-parallel computation, the stored HMM parameters μ_{11} and σ_{11} are used once. In the next M-parallel computation, the stored HMM parameters μ_{jp+1} and σ_{jp+1} are used. M output probabilities $\log b_j(\mathbf{O}_{t'+1})$, ..., and $\log b_j(\mathbf{O}_{t'+M})$ of v-th HMM are obtained by Loop A (Fig. 6). The obtained results are copied from Regω to Regδ for starting the next computation, $\log b_{j+1}(\mathbf{O}_{t'+1})$, ..., and $\log b_{j+1}(\mathbf{O}_{t'+M})$. The

Vol.2010-SLDM-144 No.8
Vol.2010-EMB-16 No.8
Vol.2010-MBL-53 No.8
Vol.2010-UBI-25 No.8
2010/3/26

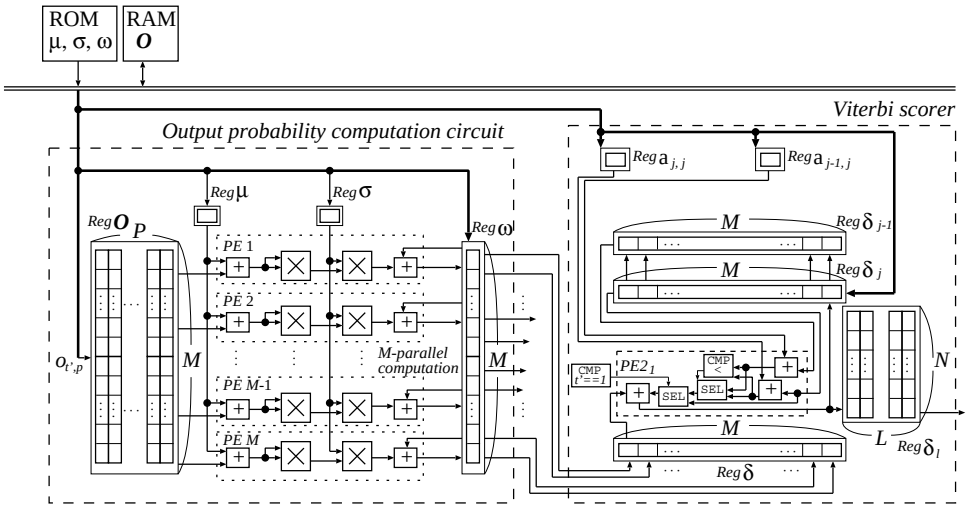

Fig. 7 FastStoreBPP VLSI architecture ($\lceil M/P \rceil = 1$).

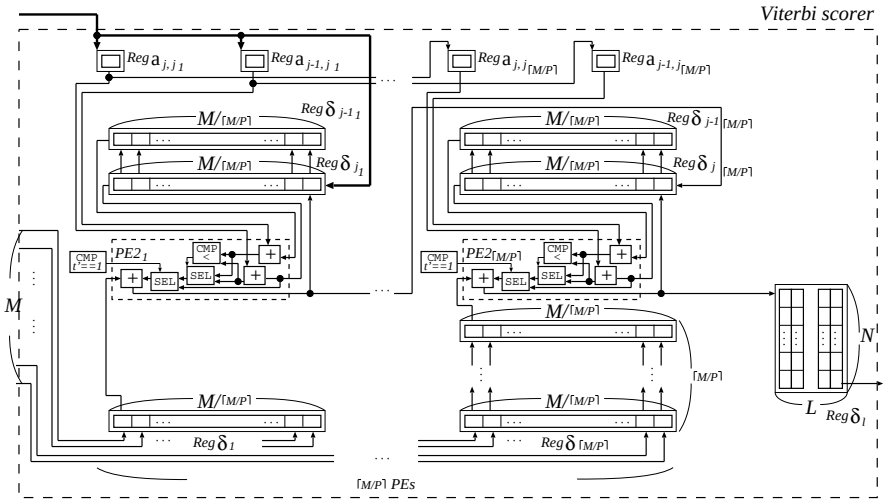

Fig. 8 Pipelined Viterbi scorer for FastStoreBPP architecture ($\lceil M/P \rceil \geq 2$).

results are fed to the Viterbi scorer. The $M \cdot N$ output probabilities of v-th HMM are obtained by Loop B (Fig. 6). $M \cdot N \cdot L$ output probabilities of HMM v', $v'+1$, ...,and $v'+L-1$ are obtained by Loop D1 (Fig. 6) with the same M input feature vectors $\mathbf{O}_{t'+1}$, ..., and $\mathbf{O}_{t'+M}$. L is the number of HMMs whose output probabilities are computed with the same input feature vectors during Loop D1. The $M \cdot N \cdot L \cdot \lceil T/M \rceil$ output probabilities of HMM v', ..., and $v'+L-1$ are obtained by Loop C1, and finally the $M \cdot N \cdot L \cdot \lceil T/M \rceil \cdot \lceil V/L \rceil$ output probabilities of all HMMs are obtained by Loop D2 (Fig. 6). The intermediate score $\log \delta'_{t'+M}(j-1)$ has to be computed during one M-parellel computation of Loop A. Consequently, we introduce pipelined Viterbi scorer for the FastStoreBPP and StoreBPP. Our pipelined Viterbi scorer for the FastStoreBPP ($M > P$) and StoreBPP ($M/2 > P$) is shown in Fig. 8. This viterbi scorer consists of $\lceil M/P \rceil$-set of $PE2s$ and register arrays for pipelined computation.

4. Evaluation

We compared the proposed FastStoreBPP, StoreBPP (Fig. 9) and StreamBPP (Fig. 10) VLSI architecture[1)–3)]. StoreBPP VLSI architecture for output probability computations and Viterbi scorer is shown in Fig. 9, where the number of $PE1s$ $M/2 \leq P$. The architecture has three register arrays, one register and $M/2$ $PE1s$ for output probability computations, and has four register arrays, two registers, and one $PE2$ for likelihood score computations based on Eq. (2), (3) and (4). Reg$\mathbf{O}$ stores M input feature vectors $\mathbf{O}_{t'+1}$, ..., and $\mathbf{O}_{t'+M}$. Regμ and Regσ store HMM parameters $-\mu_{jp}$, and σ_{jp}, respectively. Regμ has space for storing $-\mu_{jp}$ and for prestoring $-\mu_{j\,p+1}$ before the computation with $\mu_{j\,p+1}$ during the computation using μ_{jp}. Regμ is two times larger than Regσ. Regω stores HMM parameter ω_j and intermediate results. Regδ stores computed output probabilities for a Viterbi scorer. Each $PE1$ consists of two adders and two multipliers, which are used for computing Eq. (1). The architecture works as shown in the flowchart Fig. 5.

The StreamBPP (Fig. 10) architecture has two register arrays and N $PE1s$ for output probability computations. The architecture has four register arrays and N $PE2s$ for likelihood score computations based on Eq. (2), (3) and (4), where $PE2'_1$ is optimized for Eq. (2). The $PE1s$ in Figs. 10, 9 and 7 are identical. The $PE2s$ in Figs. 10, 9, 8 and

Vol.2010-SLDM-144 No.8
Vol.2010-EMB-16 No.8
Vol.2010-MBL-53 No.8
Vol.2010-UBI-25 No.8
2010/3/26

7 are identical. The StreamBPP architecture works as shown in the flowchart Fig. 3. Regμ and Regσ store HMM parameters $-\mu_{jp}$ and σ_{jp}, respectively, and Regω stores HMM parameter ω_j and intermediate results. The computation starts by reading all $2 \cdot N \cdot P + 4 \cdot N$ HMM parameters of v-th HMM from ROM and storing them to Regμ, Regσ, Regω, Regδ, Reg$\mathbf{a}_{j,j}$ and Reg$\mathbf{a}_{j-1,j}$ in Loop D (Fig. 10). For stream input o_{tp}, the intermediate results are computed with stored HMM parameters by N $PE1s$. N output probabilities $\log b_1(\mathbf{O}_t)$, ..., $\log b_N(\mathbf{O}_t)$ of the HMM are obtained by Loop A (Fig. 10). The obtained results are fed to a Viterbi scorer. $N \cdot T$ output probabilities of v-th HMM are obtained by Loop C (Fig. 10) with the same HMM parameters. The $N \cdot T \cdot V$ output probabilities of all HMMs are obtained by Loop D (Fig. 10).

Table 1 shows the register size of the FastStoreBPP, StoreBPP and StreamBPP architectures, where x_μ, x_σ, x_o , x_a , and x_f represent the bit length of μ_{jp}, σ_{jp}, o_{tp}, a_{jj}, and the output of $PE1$, respectively. N, P, and M are the number of HMM states, the dimension of input feature vector, and the number of input feature vectors in a block, respectively.

Table 2 shows the processing time for computing output probabilities of V HMMs and likelihood scores with the FastStoreBPP, StoreBPP and StreamBPP architectures, where T and L are the number of input feature vectors and the number of HMMs whose output probabilities are computed with the same input feature vectors during Loop D1 of Figs. 5 and 6, respectively.

Table 3 shows the register size, the processing time, and the number of PEs for computing output probabilities of 800 HMMs, where we assume that $N = 32$, $P = 38$, $T = 86$, $x_\mu = 8$, $x_\sigma = 8$, $x_f = 24$, $x_o = 8$,$x_a = 8$, and $V = 800$—the same values used in a recent circuit design for isolated word recognition[1),2)] and StoreBPP architecture without Viterbi scorer[5)]. We also assume that $M = 44$, $M = 29$ and $M = 44$ for one FastStoreBPP, the other FastStoreBPP and one StoreBPP architectures, respectively, where $L = 5$ for the FastStoreBPP and StoreBPP architectures. The $PE1s$ and $PE2s$ used in the FastStoreBPP, StoreBPP and StreamBPP architectures are identical. Compared with the StreamBPP and StoreBPP architectures, the FastStoreBPP has fewer registers and requires less processing time.

Compared with the FastStoreBPP ($M = 29$), the FastStoreBPP ($M = 44$) requires

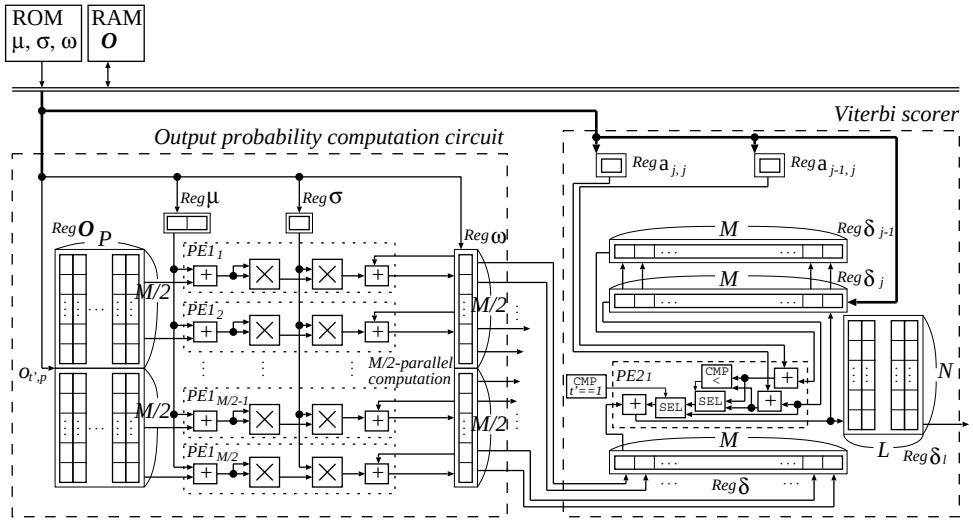

Fig. 9 StoreBPP VLSI architecture ($\lceil M/P \rceil = 2$).

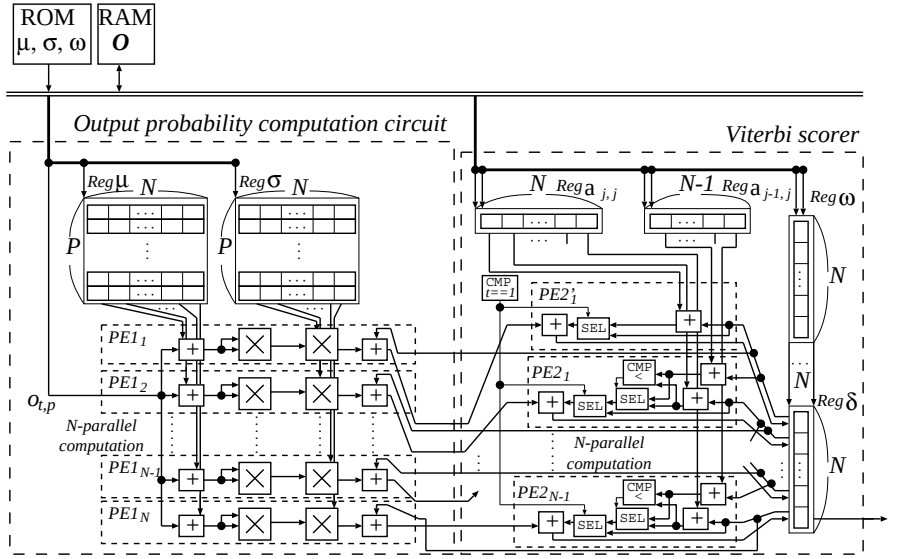

Fig. 10 StreamBPP VLSI architecture.

Vol.2010-SLDM-144 No.8
Vol.2010-EMB-16 No.8
Vol.2010-MBL-53 No.8
Vol.2010-UBI-25 No.8
2010/3/26

表 1 Register size.

	Register size (bit)
FastStoreBPP (ours)	$(P \cdot M \cdot x_o + x_\mu + x_\sigma + M \cdot x_f)$ $+(N \cdot L + 2 \cdot M + (M \cdot (\lceil M/P \rceil + 1))/2) \cdot x_f + 2 \cdot x_a$
StoreBPP (ours)	$(P \cdot M \cdot x_o + 2 \cdot x_\mu + x_\sigma + M \cdot x_f)$ $+(N \cdot L + 2 \cdot M + (M \cdot (\lceil M/P \rceil + 1))/2) \cdot x_f + 2 \cdot x_a$
StreamBPP	$(N \cdot P \cdot x_\mu + N \cdot P \cdot x_\sigma) + (3 \cdot N - 1) \cdot x_a + N \cdot x_f$

表 2 Processing times.

	Processing time (cycles)
FastStoreBPP (ours)	$\lceil V/L \rceil \cdot \{P \cdot \lceil M/2 \rceil + (2 + P) \cdot L \cdot N\} \cdot \lceil T/M \rceil$
StoreBPP (ours)	$\lceil V/L \rceil \cdot \{P \cdot M + (4 + 2 \cdot P) \cdot L \cdot N\} \cdot \lceil T/M \rceil$
StreamBPP	$V \cdot (N \cdot P + 2 \cdot N + P \cdot T)$

表 3 Evaluation of the FastStoreBPP, StoreBPP and StreamBPP performance.

	Register size (bit)	Processing time (cycles)	$\#PE1s$	$\#PE2s$
FastStoreBPP (ours)	22,000	2,315,520	44 (M=44)	2
FastStoreBPP (ours)	15,808	3,345,600	29 (M=29)	1
StoreBPP (ours)	22,008	4,631,040	22 (M=44)	1
StreamBPP	21,288	3,638,400	32	32

less processing time. Compared with the FastStoreBPP ($M = 44$), the FastStoreBPP ($M = 29$) has fewer registers and PEs. Compared with StreamBPP (Fig. 3) which requires N $PE2s$, the proposed architecture requires fewer $PE2s$ because the value of $\lceil M/P \rceil$ is at most 2 when using StoreBPP[5]. From a VLSI architectural viewpoint, the evaluation results show that full performance of the StoreBPP architecture has been exploited by a little extention of the bit length of the bus in the FastStoreBPP. From a logic design viewpoint, the register arrays of the FastStoreBPP, StoreBPP and StreamBPP architectures are designed with Flip-Flops or on-chip multi-port memories of different sizes. Data paths are designed with identical PEs, but in a different number. The control paths of these architectures are designed, as shown in the flowcharts Figs. 3, 5 and 6. The data path delay is the same for the FastStoreBPP, StoreBPP and StreamBPP designs—equal to the delay time of one $PE1$. The delay times of control paths differ between the three, but the control path delay is small compared with the data path delay.

5. Conclusions

We presented FastStoreBPP for output probability computations and Viterbi scorer and presented a new VLSI architecture. A reconfigurable architecture for both the StreamBPP and FastStoreBPP architectures are our future work.

Acknowledgment

The authors would like to thank to Assistant Professor Shingo Yoshizawa of Hokkaido University.

参 考 文 献

1) S.Yoshizawa, N.Wada, N.Hayakawa and Y.Miyanaga : Scalable Architecture for Word HMM-based Speech Recognition and VLSI Implementation in CompleteSystem, IEEE TRANS. ON CIRCUITS AND SYST., Vol.53, No.1, pp.70–77 (2006).
2) S.Yoshizawa, N.Wada, N.Hayasaka and Y Miyanaga : Scalable Architecture for Word HMM-Based Speech Recognition, Proc. of ISCAS'04, pp.417-420 (2004).
3) S.Yoshizawa, Y.Miyanaga and N.Yoshida, : On a High-Speed HMM VLSI Module with Block Parallel Processing, IEICE TRANS. Fundamentals, Vol.J85-A, No.12, pp.1440-1450 (2002).
4) Y. Kim and H. Jeong : A Systoric FPGA Architecture of Two-Level Dynamic Programming for Connected Speech Recognition, IEICE TRANS.INF & SYST., Vol.E90-D, No.2, pp.562–568 (2007).
5) K.Nakamura, M.Yamamoto, K.Takagi and N.Takagi : A VLSI Architecture for Output 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 (2010).
6) B.Mathew, A.Davis and Z.Fang, *A Low-Power Accelerator for the SPHINX 3 Speech Recognition System*, Proc. of Int'l Conf. on Compilers, Architecture and Synthesis for Embedded Systems, pp.210-219, 2003.
7) B.Mathew, A.Davis and A.Ibrahim, *Perception Coprocessors for Embedded Systems*, Proc. of ESTIMedia, pp.109-116, 2003.
8) X.Huang, F.Alleva, H.W. Hon, M.Y. Hwang, K.f. Lee and R.Rosenfeld, *The SPHINX-II speech recognition system: an overview*, Computer Speech and Language, vol.7(2), pp.137-148, 1992.