8

A VLSI Architecture for Output Probability and Likelihood Score Computations of HMM-Based Recognition Systems

Kazuhiro Nakamura[1], Ryo Shimazaki[1], Masatoshi Yamamoto[1],
Kazuyoshi Takagi[2] and Naofumi Takagi[2]
[1]*Nagoya University*
[2]*Kyoto University*
Japan

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 and continuous speech recognition), lip-reading and gesture recognition. Output probability computations (OPCs) of continuous HMMs and likelihood scorer computations (LSCs) are the most time-consuming part of HMM-based recognition systems.

High-speed VLSI architectures optimized for recognition tasks have been developed for the development of well-optimized HMM-based recognition systems (Mathew et al., 2003a;b; Nakamura et al., 2010; Yoshizawa et al., 2004; 2006; Kim & Jeong, 2007). Yoshizawa *et al*. investigated a *block-wise parallel processing* (*BPP*) for OPCs and LSCs, and proposed a high-speed VLSI architecture for word recognition (Yoshizawa et al., 2002; 2004; 2006). Nakamura *et al*. investigated a BPP, *store-based block parallel processing* (*StoreBPP*), for OPCs, and proposed a high-speed VLSI architecture for OPCs (Nakamura et al., 2010). As for OPCs and LSCs with StoreBPP, Viterbi scorer for the StoreBPP architecture is required, but not presented yet. An easy application of a Viterbi scorer to the StoreBPP architecture requires many registers and reduces the advantage of using StoreBPP. Different BPPs require different architectures of Viterbi scorer. Viterbi scorer which is suitable for StoreBPP is required for the development of well-optimized future HMM-based recognition systems.

In this chapter, we firstly show *fast store-based block parallel processing* (*FastStoreBPP*) for OPCs and LSCs, and present a Viterbi scorer which supports FastStoreBPP. FastStoreBPP exploits full performance of StoreBPP by doubling the bit length of the input to OPCs and LSCs, e.g., from 8-bit to 16-bit. We demonstrate a high-speed VLSI architecture that supports FastStoreBPP. We secondly show *multiple store-based block parallel processing* (*MultipleStoreBPP*) for OPCs and LSCs, and present a Viterbi scorer which supports MultipleStoreBPP. MultipleStoreBPP has high performance scalability by further extending the bit length of the input to OPCs and LSCs, e.g., from 8-bit to 32-bit.

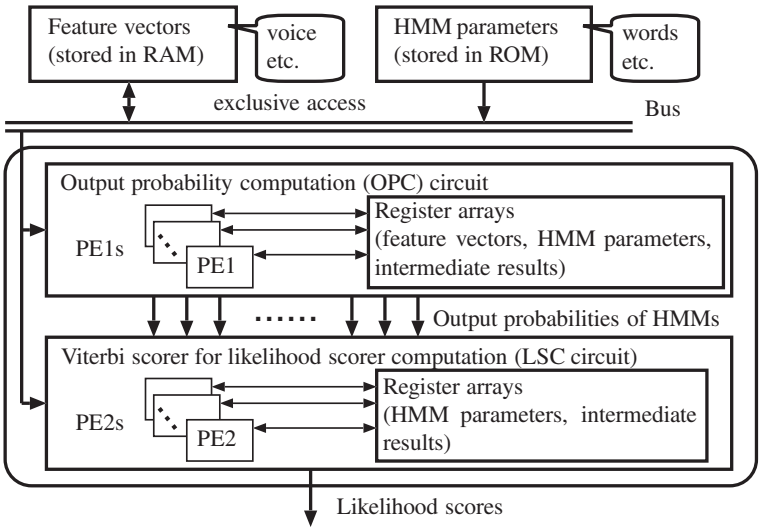


Fig. 1. Basic structure of HMM-based recognition hardware.

Compared with the StreamBPP (Yoshizawa et al., 2002; 2004; 2006) architecture, our FastStoreBPP and MultipleStoreBPP architectures have fewer registers and requires less processing time. From a VLSI architectural viewpoint, a comparison shows the efficiency of the MultipleStoreBPP architecture through its efficient use of processing elements (*PE*s).

The remainder of this chapter is organized as follows: the structure of HMM-based recognition systems is described in Section 2, the FastStoreBPP architecture is introduced in Section 3, the MultipleStoreBPP architecture is introduced in Section 4, the architectures are evaluated in Section 5, and conclusions are presented in Section 6.

2. HMM-based recognition systems

2.1 HMM-based recognition hardware

Figure 1 shows the basic structure of the relevant part of HMM-based recognition hardware (Mathew et al., 2003a;b; Nakamura et al., 2010; Yoshizawa et al., 2002; 2004; 2006; Kim & Jeong, 2007). The OPC circuit and the Viterbi scorer for LSC work together as a recognition engine. The inputs to the OPC circuit are feature vectors of several dimensions and HMM parameters. These values are stored in RAM and ROM respectively, as shown in Fig. 1. The RAM, ROM, OPC circuit and Viterbi scorer interconnect via a single bus, and memory accesses are exclusive. The OPC circuit produces HMM output probabilities. The inputs to the Viterbi scorer are these results and HMM parameters. The Viterbi scorer computes likelihood scores using the Viterbi algorithm. In HMM-based recognition systems, the most time-consuming task is OPCs and LSCs, and the OPC circuit and the Viterbi scorer accelerate these computations. The OPC circuit and the Viterbi scorer have several register arrays and *PE*s for efficient high-speed parallel processing. Feature vectors, HMM parameters and intermediate results are effectively shared between *PE*s as shown in Fig. 1. More details can be found in (Nakamura et al., 2010; Yoshizawa et al., 2002; 2004; 2006).

2.2 OPC of HMMs and LSC 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 $\mathbf{O}_t = (o_{t1}, o_{t2}, ..., o_{tP})$, $1 \leq t \leq T$. T is the number of input feature vectors, and P is the dimension of the input feature vector. For any input feature vector $\mathbf{O}_t$, the output probability of N-state continuous HMMs at the j-th state is given by

$$\log b_j(\mathbf{O}_t) = \omega_j + \sum_{p=1}^{P} \sigma_{jp}(o_{tp} - \mu_{jp})^2 \tag{1}$$

where ω_j, σ_{jp} and μ_{jp} are the parameters of the Gaussian probability density function which are precomputed and stored in ROM. The OPC circuit computes $\log b_j(\mathbf{O}_t)$ based on Eq. (1), where $1 \leq j \leq N$ and $1 \leq t \leq T$. 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 isolated word recognition systems, T, N, P, and V are 86, 32, 38, and 800, respectively (Yoshizawa et al., 2004; 2006), and for another word recognition system, T, N, P, and V are 89, 12, 16 and 100 (Yoshizawa et al., 2002).

For output probabilitie $\log b_j(\mathbf{O}_t)$, where $1 \leq j \leq N$ and $1 \leq t \leq T$, the log-likelihood score $\log S^*$ is given by

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

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

$$\log S^* = \min_{1 \leq j \leq N} [\log \delta_T(j)]. \tag{4}$$

All HMM parameters $\log \pi_j$, $\log a_{j-1,j}$, and $\log a_{j,j}$ are stored in ROM, and the Viterbi scorer computes $\log \delta_t(j)$ based on Eqs. (2) and (3). A flowchart of OPCs and LSCs is shown in Fig. 2 (Yoshizawa et al., 2004; 2006). All HMM output probabilities are obtained by $P \cdot N \cdot T \cdot V$ times the partial computation of $\log b_j(\mathbf{O}_t)$ calls. Partial computation of $\log b_j(\mathbf{O}_t)$ performs four arithmetic operations, an addition, a subtraction and two multiplications in Eq. (1). All likelihood scores are obtained by $N \cdot T \cdot V$ times the partial computation of $\log \delta_t(j)$ calls. Partial computation of $\log \delta_t(j)$ performs three additions in Eq. (3). The OPC circuit and the Viterbi scorer accelerate these computations. More details can be found in (Yoshizawa et al., 2002; 2004; 2006).

2.3 Block parallel processing for OPCs and LSCs

BPP for OPCs and LSCs was proposed as an efficient high-speed parallel processing method for HMM-based isolated word speech recognition (Yoshizawa et al., 2002). In BPP, the set of input feature vectors is called a *block*, and HMM parameters are effectively shared between the different input feature vectors for OPC. In recent years, two types of BPP are classified according to input data flow: *StreamBPP* and *StoreBPP* (Nakamura et al., 2010).

A block can be considered as a $M \cdot P$ matrix whose elements are $o_{t'p}$, where $1 \leq t' \leq M\ (\leq T)$ and $1 \leq p \leq P$. *StreamBPP* performs arithmetic operations on the input stream o_{11}, o_{12}, ..., o_{1P}, o_{21}, o_{22}, ..., o_{2P}, ..., o_{M1}, o_{M2}, ..., o_{MP} (Yoshizawa et al., 2006). StreamBPP performs N OPCs in parallel with N *PE1*s (Fig. 1) and obtains N output probabilities $\log b_1(\mathbf{O}_t)$, $\log b_2(\mathbf{O}_t)$, ..., and $\log b_N(\mathbf{O}_t)$ simultaneously. These N output probabilities are obtained every P clock cycles

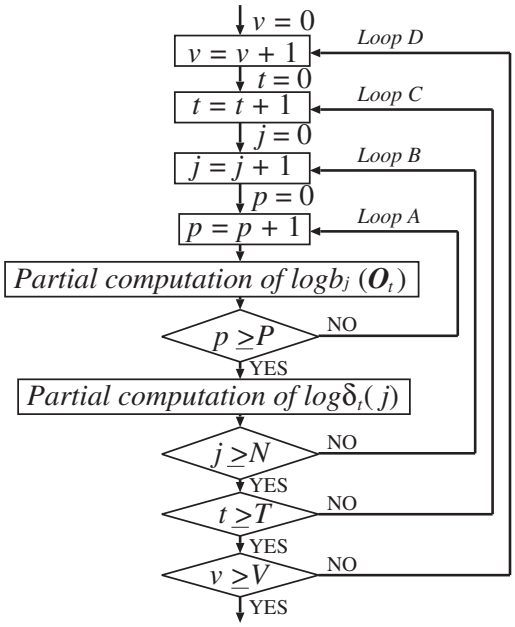


Fig. 2. Flowchart of OPCs and LSCs.

and they are fed to the Viterbi scorer for LSCs. An Viterbi scorer that supports the OPCs and LSCs in StreamBPP was presented in (Yoshizawa et al., 2006), where N LSCs are performed with N *PE2*s (Fig. 1). In the Viterbi scorer, N intermediate scores $\log \delta_t(1)$, $\log \delta_t(2)$, ..., and $\log \delta_t(N)$ are computed simultaneously.

A block can be considered as a set of M input feature vectors whose elements are $\mathbf{O}_{t'}$ where $1 \leq t' \leq M\ (\leq T)$. *StoreBPP* performs arithmetic operations to locally stored input feature vectors $\mathbf{O}_1$, $\mathbf{O}_2$, ..., and $\mathbf{O}_M$ (Nakamura et al., 2010). StoreBPP performs $\lceil M/2 \rceil$ OPCs in parallel with $\lceil M/2 \rceil$ *PE1*s (Fig. 1) and obtains M HMM output probabilities $\log b_j(\mathbf{O}_{t'+1})$, $\log b_j(\mathbf{O}_{t'+2})$, ..., and $\log b_j(\mathbf{O}_{t'+M})$. These M HMM output probabilities are obtained every $2 \cdot P$ clock cycles and they are fed to the Viterbi scorer for LSCs. Different BPPs require different Viterbi scorer architectures. In StoreBPP, a Viterbi scorer that supports the OPCs, where M HMM output probabilities are computed simultaneously, is required, but the Viterbi scorer for the StoreBPP was not addressed in (Nakamura et al., 2010). An easy introduction of the Viterbi scorer to StoreBPP requires many registers, which reduces the advantage of the StoreBPP architecture.

3. VLSI architecture for OPCs and LSCs with fast store-based block parallel processing

3.1 Fast store-based block parallel processing

A two-step process was adopted in StoreBPP to compute M HMM output probabilities $\log b_j(\mathbf{O}_{t'+1})$, $\log b_j(\mathbf{O}_{t'+2})$, ..., and $\log b_j(\mathbf{O}_{t'+M})$, where half of the output probabilities are computed simultaneously with $\lceil M/2 \rceil$ *PE1*s (Nakamura et al., 2010). The $\lceil M/2 \rceil$

computations which are performed in parallel and a ROM access are performed simultaneously in StoreBPP, where two HMM parameters $-\mu_{j,p+1}$ and $\sigma_{j,p+1}$ are required for next OPC. Because it takes two cycles to read two HMM parameters from ROM, it was appropriate to use a two-step process in StoreBPP.

StoreBPP performs $\lceil M/2 \rceil$ OPCs in parallel by using a register array of size M, where M is the size of block (Nakamura et al., 2010). We improve StoreBPP by reducing the required register size for performing M OPCs in parallel. We modify the parallel computation of $\lceil M/2 \rceil$ OPCs by doubling the bit length of the input to OPC. By this bit length extension, two HMM parameters can be read simultaneously. We call the modified parallel processing *fast store-based block parallel processing* (*FastStoreBPP*), and we show a pipelined Viterbi scorer that supports FastStoreBPP. It performs M OPCs in parallel by using a register array of size M.

A flowchart of our FastStoreBPP is shown in Fig. 3. The flowchart consists of two tasks, *M-parallel OPC* and $\lceil M/P \rceil$*-stage pipelined LSC*. In Fig. 3, the M-parallel OPC and $\lceil M/P \rceil$-stage pipelined LSC are framed by dashed and double dashed lines, respectively. The M-parallel OPC performs M OPCs in parallel with M $PE1$s. The $\lceil M/P \rceil$-stage pipelined LSC performs LSC in serial with $\lceil M/P \rceil$ $PE2$s and $\lceil M/P \rceil$ registers. Loops A′, B′, and D′ correspond to Loops A, B, and D (Fig. 2), where M HMM output probabilities and M intermediate scores are computed with M $PE1$s and $\lceil M/P \rceil$ $PE2$s. Loop C1 is based on StoreBPP, where Loop C (Fig. 2) is partially expanded in Fig. 3.

$PE1_1$, $PE1_2$, ..., and $PE1_M$ (Fig. 3) represent M processing elements which perform partial computations $\log b_j(\mathbf{O}_{t'+1})$, $\log b_j(\mathbf{O}_{t'+2})$, ..., and $\log b_j(\mathbf{O}_{t'+M})$ using Eq. (1). M HMM output probabilities $\log b_j(\mathbf{O}_{t'+1})$, $\log b_j(\mathbf{O}_{t'+2})$, ..., and $\log b_j(\mathbf{O}_{t'+M})$ are simultaneously obtained every P cycles by Loop A′. The modified M OPCs are performed in parallel and a ROM access $-\mu_{j,p+1}$ and $\sigma_{j,p+1}$ are performed simultaneously in Loop A′. The two HMM parameters are read from ROM simultaneously. These values are needed for next computation in Loop A′.

Next, the M HMM output probabilities $\log b_j(\mathbf{O}_{t'+1})$, $\log b_j(\mathbf{O}_{t'+2})$, ..., and $\log b_j(\mathbf{O}_{t'+M})$ are fed to $\lceil M/P \rceil$-stage pipelined LSC (Fig. 3) which is a new Viterbi scorer introduced for FastStoreBPP. $PE2$s represent $\lceil M/P \rceil$ processing elements for computing M intermediate scores $\log \delta_{t'+1}(j)$, $\log \delta_{t'+2}(j)$, ..., and $\log \delta_{t'+M}(j)$ using Eqs. (2) and (3). In our FastStoreBPP, Loops E_1, E_2, ..., and $\mathrm{E}_{\lceil M/P \rceil}$ (Fig. 3) perform LSC, where the i-th intermediate score $\log \delta_{t'+i}(j)$ is computed by Loop $\mathrm{E}_{\lceil i/P \rceil}$. Each Loop $\mathrm{E}_{i'}$, where $1 \leq i' \leq \lceil M/P \rceil - 1$, has a $PE2$ and computes P intermediate scores sequentially, where P clock cycles are required for the computation. Loop $\mathrm{E}_{\lceil M/P \rceil}$ has a $PE2$ and computes $M - P \cdot (\lceil M/P \rceil - 1)$ intermediate scores sequentially by using $M - P \cdot (\lceil M/P \rceil - 1)$ clock cycles. All Loop Es are pipelined in FastStoreBPP, where the last intermediate score $\log \delta_{t'+i' \cdot P}(j)$, obtained by Loop $\mathrm{E}_{i'}$, $1 \leq i' \leq \lceil M/P \rceil - 1$, is fed to the next stage Loop $\mathrm{E}_{i'+1}$. Loop A′ and Loop $\mathrm{E}_{i'}$s, $1 \leq i' \leq \lceil M/P \rceil$, proceed simultaneously, where each Loop $\mathrm{E}_{i'}$ finishes its computation before the next M output probabilities are obtained by Loop A′.

3.2 FastStoreBPP architecture for OPCs and LSCs

Our FastStoreBPP architecture that supports FastStoreBPP is shown in Fig. 4, where we assume $M \leq P$ and hence $\lceil M/P \rceil = 1$. The FastStoreBPP architecture consists of an OPC circuit and a Viterbi scorer. The architecture has two register arrays (Reg$\mathbf{O}$ and Regω),

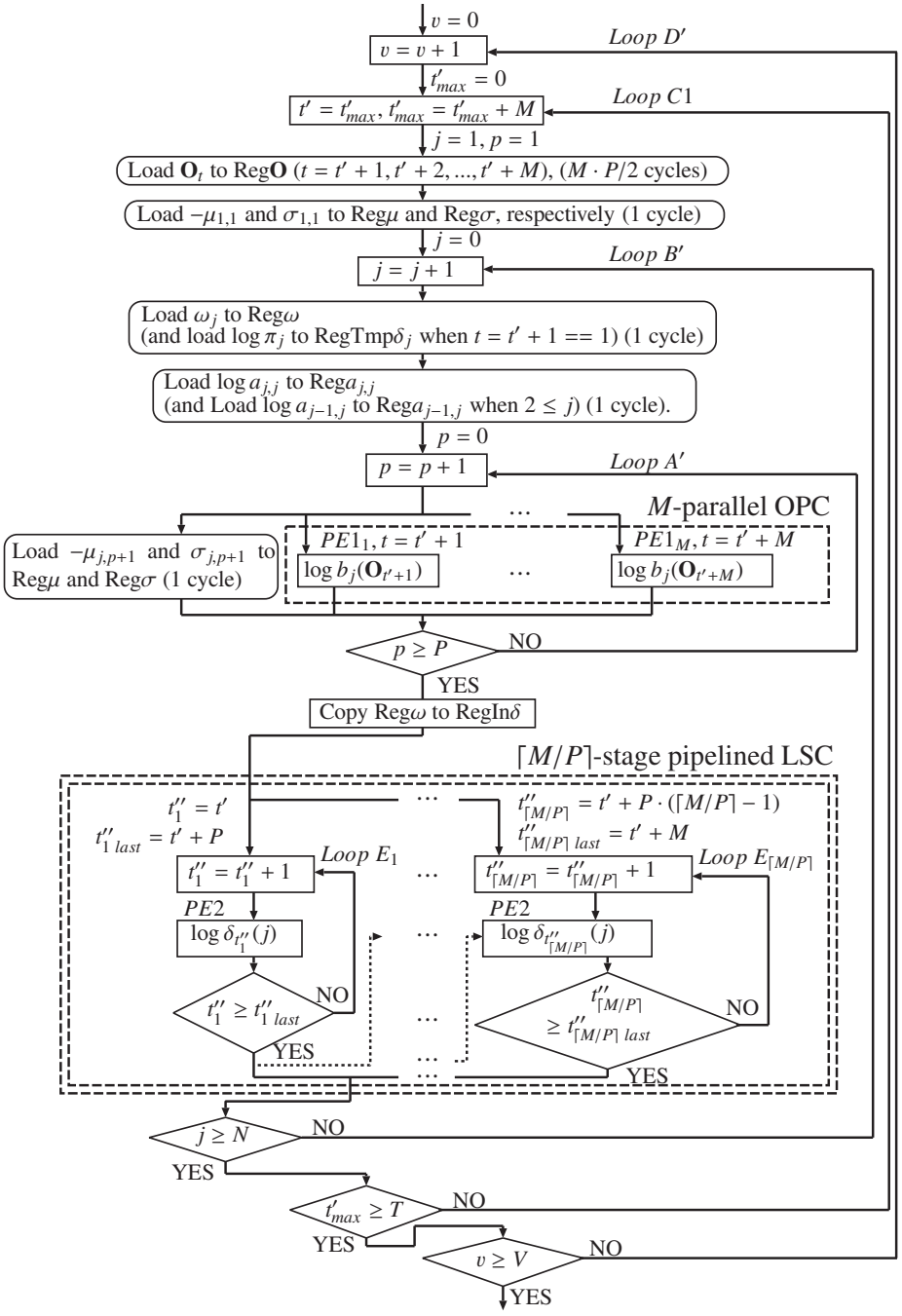


Fig. 3. Flowchart of OPCs and LSCs with FastStoreBPP.

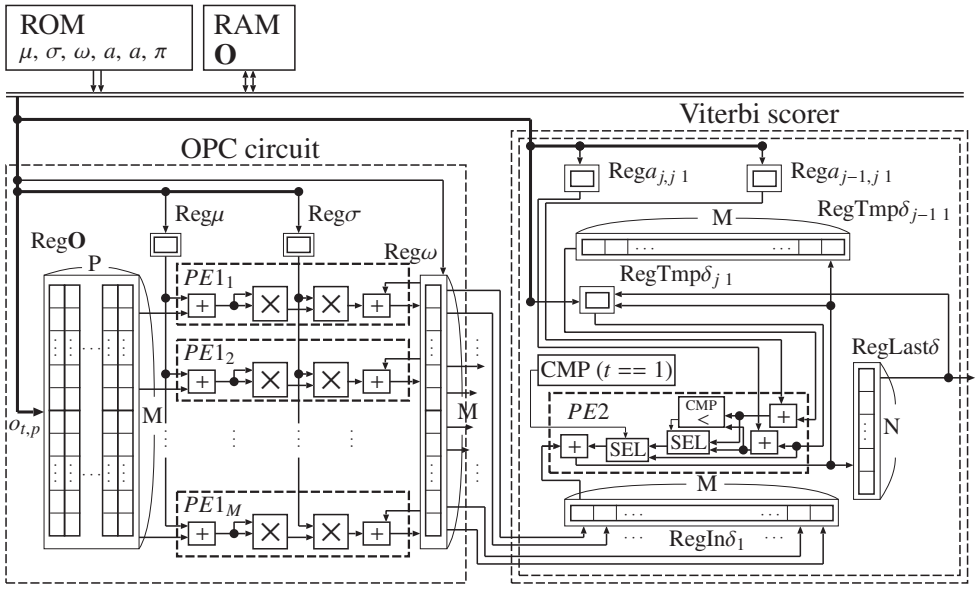


Fig. 4. FastStoreBPP architecture for OPCs and LSCs ($M \leq P$).

two registers (Regμ and Regσ), and M PE1s for OPCs. Each PE1 consists of two adders and two multipliers, which are used for computing $\omega_j + \sum_{p=1}^{P} \sigma_{jp}(o_{tp} - \mu_{jp})^2$. $PE1_i$ in the FastStoreBPP architecture, the StreamBPP architecture (Yoshizawa et al., 2006), and the StoreBPP architecture (Nakamura et al., 2010) are identical but differ in number. In addition, the architecture has three register arrays (RegInδ_1, RegLastδ, and RegTmp$\delta_{j-1\,1}$), three registers (Reg$a_{j,j\,1}$, Reg$a_{j-1,j\,1}$, and RegTmp$\delta_{j\,1}$), and a PE2 for LSCs. PE2 consists of three adders, two selectors and two comparators, which are used for LSC based on Eqs. (2) and (3). PE2 in the FastStoreBPP architecture and the StreamBPP architecture (Yoshizawa et al., 2006) are identical but differ in number.

OPC starts by reading M input feature vectors $\mathbf{O}_{t'+1}$, $\mathbf{O}_{t'+2}$, ..., and $\mathbf{O}_{t'+M}$ from RAM and storing them in Reg$\mathbf{O}$ in OPC circuit (Fig. 4) based on Loop C1 (Fig. 3). $M \cdot P/2$ cycles are required for reading M input feature vectors. Then, the HMM parameters of v-th HMM are read from ROM, which are $-\mu_{11}$, σ_{11}, and ω_1, and stored in Regμ, Regσ, and Regω, respectively, based on Loop C1 and Loop B′ (Fig. 3). For the stored input feature vectors, M intermediate results of M OPCs are simultaneously computed with the stored HMM parameters by using M PE1s (Fig. 4) based on Loop A′ (Fig. 3).

The stored HMM parameters are shared by all PE1s, and the obtained M intermediate results are stored in Regω. At the same time, two HMM parameters $-\mu_{jp+1}$ and σ_{jp+1} of v-th HMM are read from ROM and stored in Regμ and Regσ, respectively, where the values are overwritten. The HMM parameters are used in next M computations which are performed in parallel with M PE1s based on Loop A′ (Fig. 3).

M HMM output probabilities are simultaneously obtained every P cycles by M PE1s, which are $\log b_j(\mathbf{O}_{t'+1})$, $\log b_j(\mathbf{O}_{t'+2})$, ..., and $\log b_j(\mathbf{O}_{t'+M})$ of v-th HMM based on Loop A′ (Fig. 3).

The results are copied from Regω to RegInδ_1 to start LSCs $\log \delta_{t''+1}(j)$, $\log \delta_{t''+1}(j)$, ..., and $\log \delta_{t''+M}(j)$ and the next M OPCs for the $(j+1)$-th state of v-th HMM $\log b_{j+1}(\mathbf{O}_{t'+1})$, $\log b_{j+1}(\mathbf{O}_{t'+2})$, ..., and $\log b_{j+1}(\mathbf{O}_{t'+M})$ based on Loop B′ (Fig. 3). $M \cdot N$ HMM output probabilities of v-th HMM are obtained by Loop B′ (Fig. 3). $M \cdot N \cdot T$ HMM output probabilities of v-th HMM are obtained by Loop C1 (Fig. 3). $M \cdot N \cdot T \cdot V$ HMM output probabilities of all HMM are obtained by Loop D′ (Fig. 3).

Viterbi scorer, denoted by double -dashed lines in (Fig. 4), performes $\lceil M/P \rceil$-stage pipelined LSC, denoted by double -dashed lines in Fig. 3. LSC starts by reading HMM parameters of v-th HMM, $\log \pi_1$ and $\log a_{1,1}$, from ROM and storing them in RegTmp$\delta_{j\,1}$ and Reg$a_{j,j\,1}$, respectively based on Loop B′ (Fig. 3). Then, an intermediate score $\log \delta_1(1)$ is computed by $PE2$ with the HMM parameter, $\log \pi_1$, and the HMM output probability $\log b_1(\mathbf{O}_1)$ obtained using Eq. (1). The obtained intermediate score is stored in both RegTmp$\delta_{j\,1}$ and RegTmp$\delta_{j-1\,1}$ (Fig. 4). RegTmp$\delta_{j\,1}$ stores an intermediate score that is needed in the next computation in Loop E_1 (Fig. 3). RegTmp$\delta_{j-1\,1}$ stores M intermediate scores $\log \delta_{t'+1}(j)$, $\log \delta_{t'+2}(j)$, ..., and $\log \delta_{t'+M}(j)$, which is needed in the next LSC for the $(j+1)$-th state of the HMM in Loop B′. After the computation of $\log \delta_1(1)$, $M-1$ intermediate scores $\log \delta_2(1)$, $\log \delta_3(1)$, ..., and $\log \delta_M(1)$, are sequentially computed by $PE2$ using Eq. (3). In sequential computation, the last obtained intermediate score $\log \delta_M(1)$ is stored in RegTmp$\delta_{j-1\,1}$ and RegLastδ (Fig. 4). RegLastδ stores N intermediate scores that are the last obtained intermediate scores by Loop E_1 during Loop B′ (Fig. 3). These intermediate scores are $\log \delta_{t'+M}(1)$, $\log \delta_{t'+M}(2)$, ..., and $\log \delta_{t'+M}(N)$ of v-th HMM, which are required when starting LSC with new M HMM output probabilities $\log b_j(\mathbf{O}_{t'+M+1})$, $\log b_j(\mathbf{O}_{t'+M+2})$, ..., and $\log b_j(\mathbf{O}_{t'+2\cdot M})$ at the first computation in Loop E_1 (Fig. 3), given as $\log \delta_{t'+M+1}(1)$, $\log \delta_{t'+M+1}(2)$, ..., and $\log \delta_{t'+M+1}(N)$. A required intermediate score is read from RegLastδ and is stored in RegTmp$\delta_{j\,1}$ before computation. Reg$a_{j-1,j\,1}$ (Fig. 4) stores an HMM parameter $\log a_{j-1,j}$ of v-th HMM, which is used for computing $\log \delta_t(j)$ based on Eq. (3), when $2 \leq t$ and $2 \leq j$.

Our Viterbi scorer, which support FastStoreBPP, was presented in Fig. 4, where $M \leq P$ and $\lceil M/P \rceil = 1$. Our $\lceil M/P \rceil$-stage pipelined Viterbi scorer, which supports $P < M$ and $1 < \lceil M/P \rceil$, is shown in Fig. 5. The Viterbi scorer in Fig. 4 is an instance of the generalized $\lceil M/P \rceil$-stage pipelined Viterbi scorer where $\lceil M/P \rceil = 1$. The $\lceil M/P \rceil$-stage pipelined Viterbi scorer consists of RegLastδ and $\lceil M/P \rceil$ sub Viterbi scorers. The i-th stage, i.e., i-th sub Viterbi scorer consists of two register arrays (RegInδ_i and RegTmp$\delta_{j-1\,i}$), three registers (RegTmp$\delta_{j\,i}$, Reg$a_{j,j\,i}$ and Reg$a_{j-1,j\,i}$) and a $PE2$. Each RegInδ_i consists of $i \cdot P$ registers, where $i = 1$, $i = 2$, ..., $\lceil M/P \rceil - 1$. RegIn$\delta_{\lceil M/P \rceil}$ consists of $\lceil M/P \rceil \cdot (M \bmod P)$ registers. In each RegInδ_i, rows are shifted upward every P clock cycles. Each RegTmp$\delta_{j-1\,i}$ consists of P registers, where $i = 1$, $i = 2$, ..., and $\lceil M/P \rceil - 1$. RegTmp$\delta_{j-1\,\lceil M/P \rceil}$ consists of $M \bmod P$ registers. HMM parameters in Reg$a_{j,j\,i}$ and Reg$a_{j-1,j\,i}$ are copied every P clock cycles to Reg$a_{j,j\,i+1}$ and Reg$a_{j-1,j\,i+1}$, respectively, where $i = 1$, $i = 2$, ..., and $\lceil M/P \rceil - 1$. The last obtained intermediate score by $PE2$ based on Loop E_i (Fig. 3), where $i = 1$, $i = 2$, ..., and $\lceil M/P \rceil - 1$, is stored in RegTmp$\delta_{j-1\,i}$ and RegTmp$\delta_{j\,i+1}$ every P clock cycles based on $\lceil M/P \rceil$-state pipelined LSC (Fig. 3). The last obtained intermediate score by $PE2$ based on Loop $E_{\lceil M/P \rceil}$ (Fig. 3) is stored in RegTmp$\delta_{j-1\,\lceil M/P \rceil}$ and RegLastδ every P clock cycles during Loop B′ (Fig. 3). The stored intermediate scores are required when starting LSC with new M output probabilities at the first computation by Loop E_1 (Fig. 3). The required intermediate score is read from RegLastδ and stored in RegTmp$\delta_{j\,1}$ before computation.

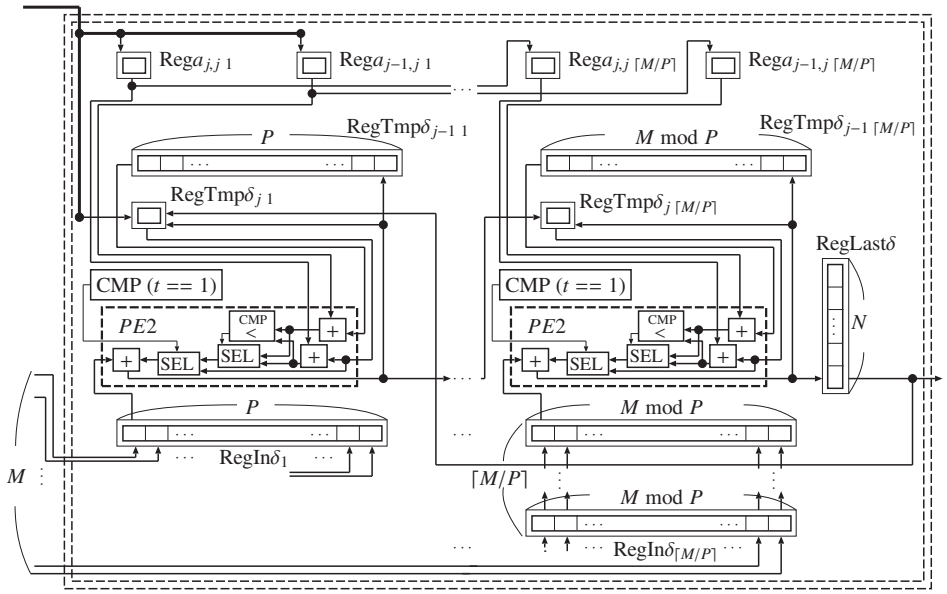


Fig. 5. Pipelined Viterbi scorer for the FastStoreBPP architecture.

4. VLSI architecture for OPCs and LSCs with multiple store-based block parallel processing

FastStoreBPP is obtained from StoreBPP by doubling the bit length of the input to OPC for reading two HMM parameters simultaneously.

We further extend the bit length of the input to OPC and we obtain *multiple store-based block parallel processing* (*MultipleStoreBPP*).

4.1 Multiple store-based block parallel processing

StoreBPP performs $M/2$ OPCs in parallel by using a register array of size M and $M/2$ PE1s, where $M/2$ OPCs are performed by a single HMM (Nakamura et al., 2010). We improve StoreBPP and further reduce the required register size for performing OPCs in parallel. We modify $M/2$-*parallel OPC*, where $M/2$ OPCs are performed in parallel, to deal with multiple HMMs by further extending the bit length of the input to OPC. By this bit-length extension, $L' \cdot M'/2$ OPCs are performed in parallel, where L' HMM parameters can be read from ROM simultaneously. We call the modified parallel processing *MultipleStoreBPP*. Our MultipleStoreBPP performs $M'/2$-*parallel OPC*, where $M'/2$ OPCs are performed in parallel, to L' HMMs and $L' \cdot M'/2$ OPCs are performed in parallel by using a register array of size M'.

A flowchart of our MultipleStoreBPP is shown in Fig. 6. Loops D2′, C1′, B″, and A″ in Fig. 6 are based on StoreBPP. In our MultipleStoreBPP, Loop D′ (Fig. 3) is partially expanded as shown in Fig. 6 for performing $L' \cdot M'/2$ OPCs in parallel in Loop A″. By the expansion, input feature vectors are effectively shared between different $M'/2$-parallel OPCs.

The flowchart consists of L' $M'/2$-parallel OPCs and L' $\lceil M'/(2 \cdot P)\rceil$-stage pipelined LSCs. Each $M'/2$-parallel OPC and $\lceil M'/(2 \cdot P)\rceil$-stage pipelined LSC are denoted by dashed and

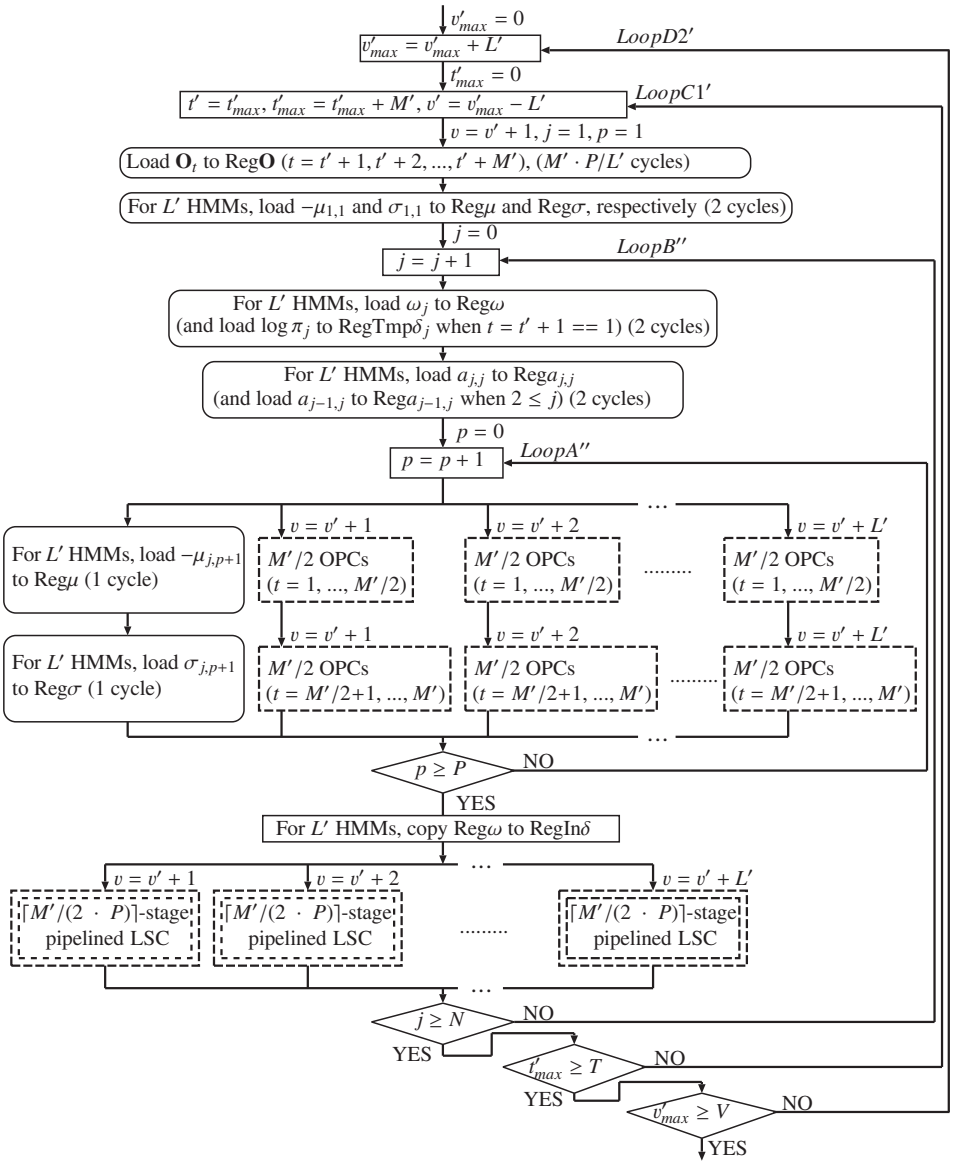


Fig. 6. Flowchart of OPCs and LSCs with MultipleStoreBPP.

double-dashed lines, respectively, in Fig. 6. Each $M'/2$-parallel OPC performs $M'/2$ OPCs in parallel with the same PE1s used in StoreBPP and FastStoreBPP, but differ in number. Each $\lceil M'/(2 \cdot P) \rceil$-stage pipelined LSC computes likelihood scores based on $\lceil M/P \rceil$-stage pipelined LSC denoted by double-dashed lines in Fig. 3 with the same PE2s used in FastStoreBPP, but differ in number. Loops A$'$ and B$'$ (Fig. 3) correspond to Loops A$''$ and B$''$, respectively. By Loop A$''$, the output probabilities of L' HMMs, $(v'+1)$-th to $(v'+L')$-th HMMs, are computed with $L' \cdot M'/2$ PE1s. These output probabilities are simultaneously obtained every P clock cycles by Loop A$''$. In Loop A$''$, firstly, L' $M'/2$-parallel OPCs and ROM access $-\mu_{j,p+1}$ of L' HMMs are performed simultaneously. Secondly, L' $M'/2$-parallel OPCs and ROM access $\sigma_{j,p+1}$ of L' HMMs are performed simultaneously. $2 \cdot L'$ HMM parameters are read from ROM using two cycles. These HMM parameters are needed for next computation in Loop A$''$. Then, the obtained $L' \cdot M'/2$ output probabilities are fed to L' LSCs, where each LSC is the same Viterbi scorer as that introduced for FastStoreBPP, as shown in Fig. 3. These L' LSCs support LSC of $(v'+1)$-th to $(v'+L')$-th HMMs. Loop A$''$ and L' LSCs proceed simultaneously.

4.2 MultipleStoreBPP architecture for OPCs and LSCs

Our MultipleStoreBPP VLSI architecture is shown in Fig. 7, where we assume $M' \leq 2 \cdot P$ and hence $\lceil M'/(2 \cdot P) \rceil = 1$. The MultipleStoreBPP architecture consists of L' OPC circuits and L' Viterbi scorers. The architecture has $2 \cdot L' + 1$ register arrays (Reg$\mathbf{O}$, Regσ and Regω), L' registers (Regμ), and $L' \cdot M'/2$ PE1s for OPCs of L' HMMs. Each PE1 consists of two adders and two multipliers, which are used for computing $\omega_j + \sum_{p=1}^{P} \sigma_{jp}(o_{tp} - \mu_{jp})^2$. PE1s in our MultipleStoreBPP and FastStoreBPP architectures, Yoshizawa et al. (2006), and Nakamura et al. (2010) are identical but differ in number. In addition, the architecture has $3 \cdot L'$ register arrays (RegInδ_1, RegLastδ, and RegTmp$\delta_{j-1\,1}$), $3 \cdot L'$ registers (Reg$a_{j,j\,1}$, Reg$a_{j-1,j\,1}$, and RegTmp$\delta_{j\,1}$), and L' PE2s for LSCs of L' HMMs. PE2 consists of three adders, two selectors and two comparators, which are used for LSC on the basis of Eqs. (2) and (3). PE2s in our MultipleStoreBPP and FastStoreBPP architectures and Yoshizawa et al. (2006) are identical but differ in number.

OPC starts by reading M' input feature vectors $\mathbf{O}_{t'+1}$, ..., and $\mathbf{O}_{t'+M'}$ from RAM and storing them in Reg$\mathbf{O}$ in OPC circuit$_1$ (Fig. 7) based on Loop C1$'$ (Fig. 6). $M' \cdot P/L'$ cycles are required for reading M' feature vectors. Then, the HMM parameters of L' HMMs, i.e., $(v'+1)$-th to $(v'+L)$-th HMMs, are read from ROM, which are $-\mu_{11}$, σ_{11} and ω_1, and stored in Regμ, Regσ, and Regω, respectively, based on Loop $C1'$ and Loop B'' (Fig. 6). For half of the stored input feature vectors, $\mathbf{O}_{t'+1}$, ..., and $\mathbf{O}_{t'+M'/2}$, $L' \cdot M'/2$ intermediate results of L' OPCs are simultaneously computed with the stored HMM parameters by using $L' \cdot M'/2$ PE1s (Fig. 7) based on Loop A$''$ (Fig. 6). Then, for the other half of the stored input feature vectors, $\mathbf{O}_{t'M'/2+1}$, ..., and $\mathbf{O}_{t'+2\cdot M'}$, $L' \cdot M'/2$ intermediate results of L' OPCs are simultaneously computed with the stored HMM parameters by using $L' \cdot M'/2$ PE1s (Fig. 7) based on Loop A$''$ (Fig. 6). The stored M' input feature vectors are effectively shared by all OPC circuits.

In each OPC circuit, denoted by dashed line in Fig. 7, the two HMM parameters, $-\mu_{jp}$ and σ_{jp} are shared by all PE1s, and the obtained first $M'/2$ intermediate results and the second $M'/2$ intermediate results are stored in Regω using two cycles. At the same time, the two HMM parameters $-\mu_{jp+1}$ and σ_{jp+1} are read from ROM and stored in Regμ and Regσ, respectively,

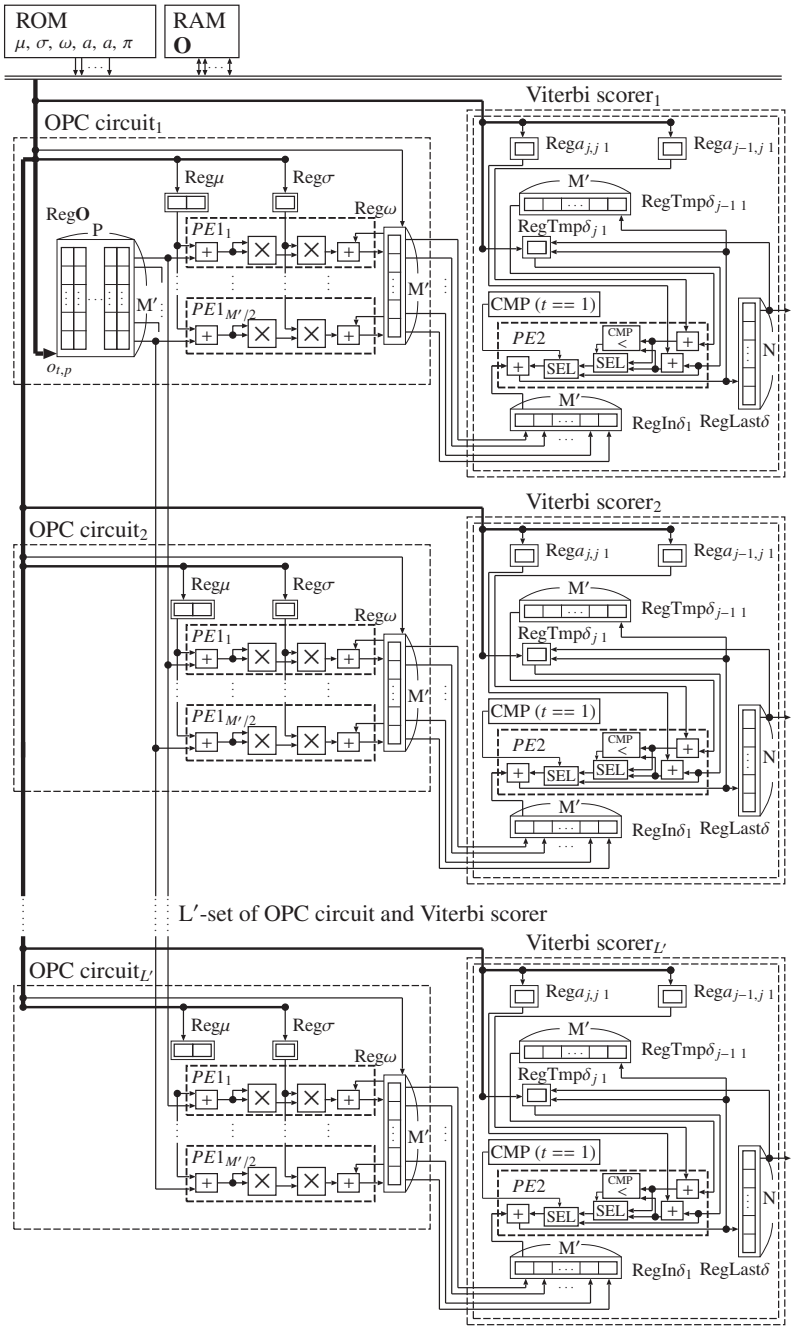


Fig. 7. MultipleStoreBPP architecture for OPCs and LSCs.

using two clock cycles. The HMM parameters are used in the next M'-parallel computation in the OPC circuit.

$L' \cdot M'$ output probabilities are simultaneously obtained every $2 \cdot P$ clock cycles by L' OPC circuits, which are $\log b_j(\mathbf{O}_{t'+1})$, ..., and $\log b_j(\mathbf{O}_{t'+M'})$ of L' HMMs based on Loop A'' (Fig. 6). The results are copied from Regω to RegInδ_1 for starting LSCs of L' HMMs and next OPCs for $(j+1)$-th states of L' HMMs $\log b_{j+1}(\mathbf{O}_{t'+1})$, ..., and $\log b_{j+1}(\mathbf{O}_{t'+M'})$ based on Loop B'' (Fig. 6). $L' \cdot M' \cdot N$ output probabilities of L' HMMs are obtained by Loop B'' with the same M' input feature vectors $\mathbf{O}_{t'+1}$, ..., and $\mathbf{O}_{t'+M'}$.

Each Viterbi scorer, denoted by double-dashed lines in Fig. 7, performs $\lceil M'/(2 \cdot P) \rceil$-stage pipelined LSC, denoted by double-dashed lines in Fig. 3. LSC starts by reading HMM parameters of L' HMMs, $\log \pi_1$ and $\log a_{1,1}$, from ROM and storing them in RegTmp$\delta_{j\,1}$ and RegTmp$a_{j,j\,1}$ based on Loop B'' (Fig. 6). Then, in each Viterbi scorer, an intermediate score $\log \delta_1(1)$ is computed by $PE2$ with the HMM parameter and the output probability obtained using Eq. (1). The obtained intermediate score is stored in both RegTmp$\delta_{j\,1}$ and RegTmp$\delta_{j-1\,1}$ (Fig. 7). RegTmp$\delta_{j\,1}$ stores an intermediate score that is needed in the next computation in Loop E_1 (Fig. 3). RegTmp$\delta_{j-1\,1}$ stores M' intermediate scores $\log \delta_{t'+1}(j)$, ..., and $\log \delta_{t'+M'}(j)$, which is needed in the next LSC for the $(j+1)$-th state of the HMM in Loop B''. After the computation of $\log \delta_1(1)$, $M'-1$ intermediate scores $\log \delta_2(1)$, ..., and $\log \delta_{M'}(1)$ are sequentially computed by $PE2$ on the basis of Eq. (3). In sequential computation, the last obtained intermediate score $\log \delta_{M'}(1)$, is stored in RegTmp$\delta_{j-1\,1}$ and RegLastδ (Fig. 7). RegLastδ stores N intermediate scores that are the last obtained intermediate scores by Loop E_1 during Loop B''. These intermediate scores are $\log \delta_{t'+M'}(1)$, ..., and $\log \delta_{t'+M'}(N)$ of v-th HMM, which are required when starting LSC with new M' output probabilities $\log b_j(\mathbf{O}_{t'+M'+1})$, ..., and $\log b_j(\mathbf{O}_{t'+2 \cdot M'})$ at the first computations in Loop E_1, given as $\log \delta_{t'+M'+1}(1)$, ..., and $\log \delta_{t'+M'+1}(N)$. A required intermediate score is read from RegLastδ and is stored in RegTmp$\delta_{j\,1}$ before computation. Reg$a_{j-1,j\,1}$ (Fig. 7) stores an HMM parameter $\log a_{j-1,j}$, which is used for computing $\log \delta_{t'}(j)$ on the basis of Eq. (3) when $2 \leq t'$ and $2 \leq j$.

Our Viterbi scorers, which support MultipleStoreBPP, were presented in Fig. 7 as Viterbi scorer$_i$, where $M' \leq 2 \cdot P$ and $\lceil M'/(2 \cdot P) \rceil = 1$. Our $\lceil M'/(2 \cdot P) \rceil$-stage pipelined Viterbi scorer, which supports $2 \cdot P < M'$ and $1 < \lceil M'/(2 \cdot P) \rceil$, is shown in Fig. 8. The Viterbi scorers shown in Fig. 7 are instances of the generalized $\lceil M'/(2 \cdot P) \rceil$-stage pipelined Viterbi scorer where $\lceil M'/(2 \cdot P) \rceil = 1$. The $\lceil M'/(2 \cdot P) \rceil$-stage pipelined Viterbi scorer consists of a register array RegLastδ and $\lceil M'/(2 \cdot P) \rceil$ sub Viterbi scorers. The i-th stage, i.e., i-th sub Viterbi scorer consists of two register arrays (RegInδ_i and RegTmp$\delta_{j-1\,i}$), three registers (RegTmp$\delta_{j\,i}$, Reg$a_{j,j\,i}$, and Reg$a_{j-1,j\,i}$), and a $PE2$. Each RegInδ_i consists of $i \cdot P$ registers, where $i = 1$, ..., and $\lceil M'/(2 \cdot P) \rceil - 1$. RegIn$\delta_{\lceil M'/(2 \cdot P) \rceil}$ consists of $\lceil M'/(2 \cdot P) \rceil \cdot (M' \bmod (2 \cdot P))$ registers.

In each RegInδ_i, rows are shifted upward every P clock cycles. Each RegTmp$\delta_{j-1\,i}$ consists of P registers, where $i = 1$, ..., and $\lceil M'/(2 \cdot P) \rceil - 1$. RegTmp$\delta_{j-1\,\lceil M'/(2 \cdot P) \rceil}$ consists of $M' \bmod (2 \cdot P)$ registers. HMM parameters in Reg$a_{j,j\,i}$ and Reg$a_{j-1,j\,i}$ are copied every P cycles to Reg$a_{j,j\,i+1}$ and Reg$a_{j-1,j\,i+1}$, respectively, where $i = 1$, ..., $\lceil M'/P \rceil - 1$. The last obtained intermediate score by $PE2$ based on Loop E_i (Fig. 3), where $i = 1$, ..., and $\lceil M'/(2 \cdot P) \rceil - 1$, is stored in RegTmp$\delta_{j-1\,i}$ and RegTmp$\delta_{j\,i+1}$ every P cycles based on $\lceil M'/(2 \cdot P) \rceil$-stage pipelined LSC (Fig. 3). The last obtained intermediate score by $PE2$ based

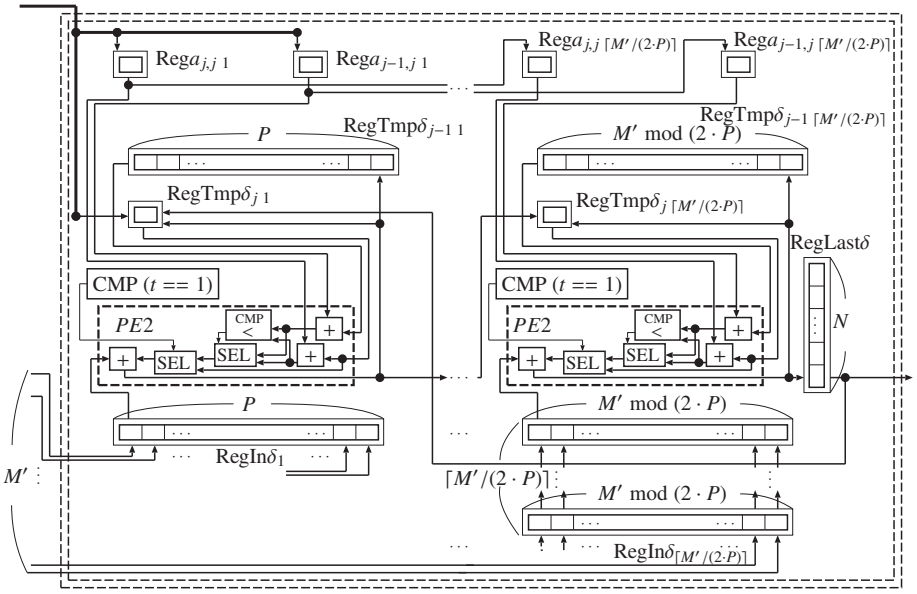


Fig. 8. Pipelined Viterbi scorer for the MultipleStoreBPP architecture (Viterbi scorer$_i$, $i = 1, ..., L'$).

on Loop E$_{\lceil M'/(2\cdot P)\rceil}$ (Fig. 3) is stored in RegTmp$\delta_{j-1\ \lceil M'/(2\cdot P)\rceil}$ and RegLastδ every P cycles during Loop B$''$ (Fig. 6). The stored intermediate scores are required when starting LSC with new M' output probabilities for first computation in Loop E$_1$ (Fig. 3). The required intermediate score is read from RegLastδ and stored in RegTmp$\delta_{j\ 1}$ before computations.

5. Evaluation

We compared StreamBPP Yoshizawa et al. (2004; 2006), StoreBPP Nakamura et al. (2010), FastStoreBPP (Figs. 3, 4, and 5), and MultipleStoreBPP (Figs. 6, 7, and 8) VLSI architectures.

Table 1 shows the register size of MultipleStoreBPP, FastStoreBPP, StoreBPP, and StreamBPP architectures, where x_μ, x_σ, x_ω, x_o , x_a, and x_f represent the bit length of μ_{jp}, σ_{jp}, ω_j, o_{tp}, a_{jj}, and the output of $PE1$, respectively. N, P, and M are the number of HMM states, the dimension of the input feature vector, and the number of input feature vectors in a block, respectively. M' and L' are the number of input feature vectors in a block with MultipleStoreBPP and the number of HMMs whose output probabilities are simultaneously computed by Loop A$''$ (Fig. 6) with MultipleStoreBPP, respectively. OPC and Viterbi scorer represent the register size of the OPC circuit and Viterbi scorer, respectively.

Table 2 shows the processing time for computing output probabilities of V HMMs and likelihood scores with MultipletStoreBPP, FastStoreBPP, StoreBPP, and StreamBPP architectures, where L is the number of HMMs whose output probabilities are computed using the same input feature vectors with StoreBPP Nakamura et al. (2010). OPC and the Viterbi scorer represent the number of clock cycles for OPC and additional cycles for LSC, respectively.

MultipleStoreBPP [bit]	
OPC	$P \cdot M' \cdot x_o + (2 \cdot x_\mu + x_\sigma + M' \cdot x_f) \cdot L'$
Viterbi scorer	$L' \cdot [\{N + (2 \cdot P + 1)(\lceil (M+1)/(2 \cdot P) \rceil - 1) + 2 \cdot P \cdot \sum_{i=0}^{\lceil (M+1)/(2 \cdot P) \rceil - 1} i + (M \bmod 2 \cdot P) + 1 + (M \bmod 2 \cdot P) \cdot \lceil M/(2 \cdot P) \rceil\} \cdot x_f + 2 \cdot \lceil M/(2 \cdot P) \rceil \cdot x_a]$

FastStoreBPP [bit]	
OPC	$P \cdot M \cdot x_o + x_\mu + x_\sigma + M \cdot x_f$
Viterbi scorer	$\{N + (P+1)(\lceil (M+1)/P \rceil - 1) + P \cdot \sum_{i=0}^{\lceil (M+1)/P \rceil - 1} i + (M \bmod P) + 1 + (M \bmod P) \cdot \lceil M/P \rceil\} \cdot x_f + 2 \cdot \lceil M/P \rceil \cdot x_a$

StreamBPP Yoshizawa et al. (2006) [bit]	
OPC	$N \cdot P \cdot x_\mu + N \cdot P \cdot x_\sigma + N \cdot x_f$
Viterbi scorer	$(2 \cdot N - 1) \cdot x_a + N \cdot x_\omega + N \cdot x_f$

StoreBPP Nakamura et al. (2010) [bit]	
OPC	$P \cdot M \cdot x_o + 2 \cdot x_\mu + x_\sigma + M \cdot x_f$
Viterbi scorer	— not available

Table 1. Register size.

MultipleStoreBPP [cycle]	
OPC	$\lceil V/L' \rceil \{\lceil P \cdot M'/L' \rceil + (1 + 2 \cdot P) \cdot N\} \lceil T/M' \rceil$
Viterbi scorer	$\lceil V/L' \rceil \{2 \cdot N \cdot \lceil T/M' \rceil + N\}$

FastStoreBPP [cycle]	
OPC	$V \cdot \{P \cdot \lceil M/2 \rceil + (1+P) \cdot N\} \lceil T/M \rceil$
Viterbi scorer	$V \cdot \{N \cdot \lceil T/M \rceil + N\}$

StreamBPP Yoshizawa et al. (2006)[cycle]	
OPC	$V \cdot (2 \cdot N \cdot P + N + P \cdot T)$
Viterbi scorer	$V \cdot (3 \cdot N - 1)$

StoreBPP Nakamura et al. (2010) [cycle]	
OPC	$\lceil V/L \rceil \{P \cdot M + (1 + 2 \cdot P) \cdot L \cdot N\} \lceil T/M \rceil$
Viterbi scorer	— not available

Table 2. Processing times.

Table 3 shows the register size, processing time, and the number of *PEs* for computing output probabilities of 800 HMMs and likelihood scores, where it is assumed that $N = 32$, $P = 38$, $T = 86$, $x_\mu = 8$, $x_\sigma = 8$, $x_f = 24$, $x_o = 8$, $x_a = 8$, and $V = 800$. These values are the same as those used in a recent circuit design for isolated word recognition Nakamura et al. (2010); Yoshizawa et al. (2004; 2006). In addition, we assume that $M' = 12$, $L' = 4$ for the MultipleStoreBPP architecture. Futhermore, ratios compared with StreamBPP are shown in Table 3. Compared with the StreamBPP architectures, the MultipleStoreBPP architecture has fewer registers (48% = 10,432/21,752) and requires less processing time (91% = 4,233,600/4,661,600). The number of *PE*2s in MultipleStoreBPP and FastStoreBPP are less than that in StreamBPP.

Figure 9 shows the processing time, and the number of *PE*s in MultipleStoreBPP, FastStoreBPP, and StreamBPP architectures, and the value of M, where $M = M' \cdot L'/2$ for MultipleStoreBPP.

	Reg. size [bit]	Proc. time [cycle]	#PEs	
			PE1	PE2
MultipleStoreBPP $(M', L') = (12, 4)$	10,432 (48%)	4,233,600 (91%)	24 (75%)	4 (13%)
FastStoreBPP $(M = 24)$	9,848 (45%)	5,580,800 (120%)	24 (75%)	1 (3%)
StreamBPP	21,752 (100%)	4,661,600 (100%)	32 (100%)	32 (100%)

Table 3. Evaluation of the MultipleStoreBPP, FastStoreBPP, and StreamBPP performance.

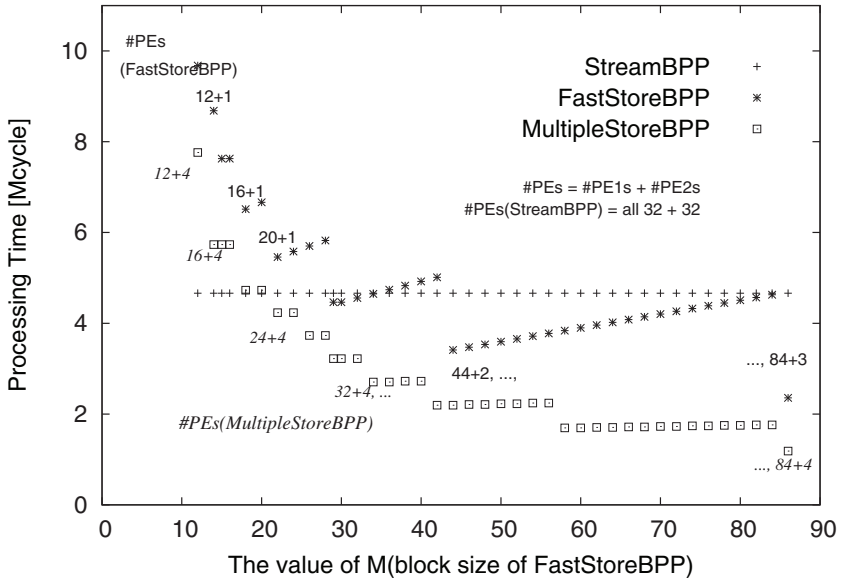


Fig. 9. Processing time, the number of PEs and value of M.

This graph shows that the processing time of MultipleStoreBPP is less than that of FastStoreBPP architecture. It is also less than that of the StreamBPP architecture when M is greater than 22.

Figure 10 shows the register size of MultipleStoreBPP, FastStoreBPP, and StreamBPP architectures as well as the value of M (block size). This graph shows that the register size of MultipleStoreBPP is less than those of FastStoreBPP, and StreamBPP architectures when M is greater than 36 and less than 74.

Table 4 shows the circuit area, clock period, and power dissipation of the OPC and LSC circuits based on the MultipleStoreBPP and FastStoreBPP architectures, which are derived from the report of the Synopsys Design Compiler (Ver. B-2008.09-SP5), where the target technology is the 90nm technology (STARC 90nm) and the report on power dissipation is obtained with report_power command after logic synthesis. In the table, the delay and area represent the

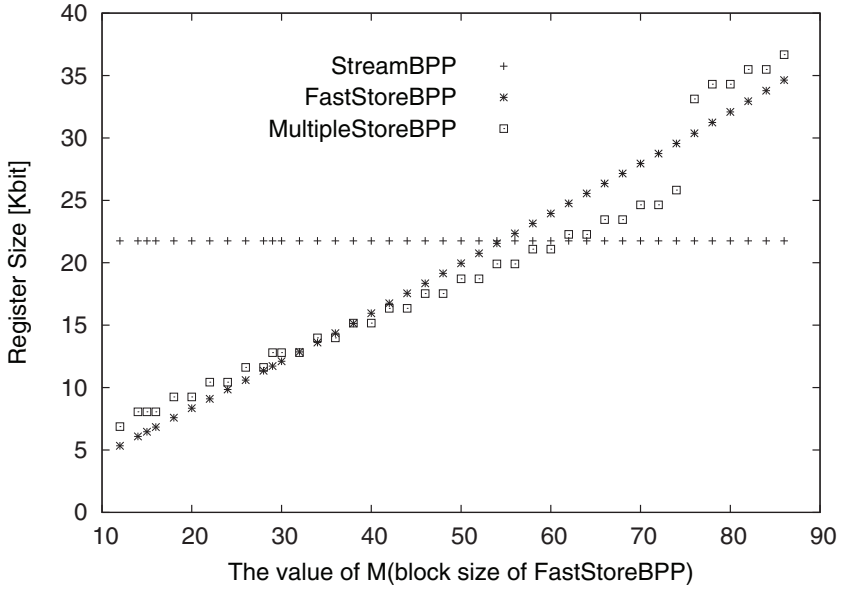


Fig. 10. Register size and the value of M.

Arch.	area [μm^2]	delay [ns]	power [mW]
MultipleStoreBPP (#$PE1 = 44,\ M' = 22,\ L' = 4$)	1,042,492	2.8	4.5
FastStoreBPP (#$PE1 = 32,\ M = 32$)	849,955	2.7	3.5
FastStoreBPP (#$PE1 = 44,\ M = 44$)	1,155,595	2.7	4.8

Table 4. Area, delay and power of OPC and LSC circuits.

minimum clock period and area of the circuit, respectively. Power represents the power dissipation of the circuit whose operating clock frequency is 11 MHz, and for an 800-word real-time isolated word recognition, recognition in 0.2 s is achieved by the MultipleStoreBPP architecture for $T = 86$, a 1-s speech, $V = 800$, $N = 32$, $P = 38$, $M' = 22$ and $L' = 4$. Compared with the FastStoreBPP architecture, the MultipleStoreBPP architecture has lower power and less area, because the MultipleStoreBPP architecture has fewer registers when the number of PE1s is 44.

6. Conclusions

We presented MultipleStoreBPP for OPCs and LSCs and presented a new VLSI architecture. MultipleStoreBPP performs parallel-OPCs and pipelined-LSCs for multiple HMMs. Compared with the conventional StoreBPP architecture, the MultipleStoreBPP architecture supports LSC. Furthermore, compared with StreamBPP and FastStoreBPP architectures, the MultipleStoreBPP architecture requires fewer registers and less processing time. In terms of the VLSI architecture the comparison shows the efficiency of the MultipleStoreBPP architecture.

7. Acknowledgements

This work is supported by the VLSI Design and Education Center (VDEC), the University of Tokyo in collaboration with Synopsys, Inc. and the Semiconductor Technology Academic Research Center (STARC).

8. References

B. Mathew; A. Davis & A. Ibrahim. (2003). Perception Coprocessors for Embedded Systems. *Proc. of ESTIMedia*, 109 – 116

B. Mathew; A. Davis & Z. Fang. (2003). A Low-Power Accelerator for the SPHINX 3 Speech Recognition System. *Proc. of Int'l Conf. on Compilers, Architecture and Synthesis for Embedded Systems*, 210 – 219

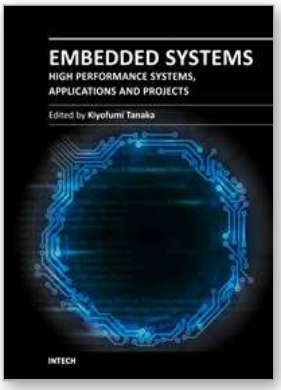
K. Nakamura; M. Yamamoto; K. Takagi & N. Takagi. (2010). A VLSI Architecture for Output Probability Computations of HMM-Based Recognition Systems with Store-Based Block Parallel Processing, *IEICE TRANS. INF. & SYST.*, Vol. E93-D, No. 2, 300 – 305

S. Yoshizawa; Y. Miyanaga & N. Yoshida. (2002). On a High-Speed HMM VLSI Module with Block Parallel Processing. *IEICE TRANS. Fundamentals*, Vol. J85-A, No. 12, 1440 – 1450

S. Yoshizawa; N. Wada; N. Hayasaka & Y. Miyanaga. (2004). Scalable Architecture for Word HMM-Based Speech Recognition. *Proc. of ISCAS'04*, 417 – 420

S. Yoshizawa; N. Wada; N. Hayakawa & Y. Miyanaga. (2006). Scalable Architecture for Word HMM-based Speech Recognition and VLSI Implementation in Complete System. *IEEE TRANS. ON CIRC. & SYST.*, Vol. 53, No. 1, 70 – 77

Y. Kim & H. Jeong. (2007). A Systolic FPGA Architecture of Two-Level Dynamic Programming for Connected Speech Recognition. *IEICE TRANS. INF. & SYST.*, Vol. E90-D, No. 2, 562 – 568



Embedded Systems - High Performance Systems, Applications and Projects

Edited by Dr. Kiyofumi Tanaka

ISBN 978-953-51-0350-9
Hard cover, 278 pages
Publisher InTech
Published online 16, March, 2012
Published in print edition March, 2012

Nowadays, embedded systems - computer systems that are embedded in various kinds of devices and play an important role of specific control functions, have permeated various scenes of industry. Therefore, we can hardly discuss our life or society from now onwards without referring to embedded systems. For wide-ranging embedded systems to continue their growth, a number of high-quality fundamental and applied researches are indispensable. This book contains 13 excellent chapters and addresses a wide spectrum of research topics of embedded systems, including parallel computing, communication architecture, application-specific systems, and embedded systems projects. Embedded systems can be made only after fusing miscellaneous technologies together. Various technologies condensed in this book as well as in the complementary book "Embedded Systems - Theory and Design Methodology", will be helpful to researchers and engineers around the world.

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821