IEICE TRANS. INF. & SYST., VOL.E93–D, NO.2 FEBRUARY 2010

PAPER

A VLSI Architecture for Output Probability Computations of HMM-Based Recognition Systems with Store-Based Block Parallel Processing*

Kazuhiro NAKAMURA[†a)], ***Member***, **Masatoshi YAMAMOTO**[†], ***Nonmember***, **Kazuyoshi TAKAGI**[†], ***and*** **Naofumi TAKAGI**[†], ***Members***

SUMMARY In this paper, a fast and memory-efficient VLSI architecture for output probability computations of continuous Hidden Markov Models (HMMs) is presented. These computations are the most time-consuming part of HMM-based recognition systems. High-speed VLSI architectures with small registers and low-power dissipation are required for the development of mobile embedded systems with capable human interfaces. We demonstrate store-based block parallel processing (StoreBPP) for output probability computations and present a VLSI architecture that supports it. When the number of HMM states is adequate for accurate recognition, compared with conventional stream-based block parallel processing (StreamBPP) architectures, the proposed architecture requires fewer registers and processing elements and less processing time. The processing elements used in the StreamBPP architecture are identical to those used in the StoreBPP architecture. From a VLSI architectural viewpoint, a comparison shows the efficiency of the proposed architecture through efficient use of registers for storing input feature vectors and intermediate results during computation.

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

1. Introduction

Mobile embedded systems with sophisticated natural human interfaces, such as speech recognition, lip reading, and gesture recognition, are required for the realization of future ubiquitous computing.

Recognition tasks can be implemented either on processors (CPUs and DSPs) or dedicated hardware (ASICs). Although processor-based approaches offer flexibility, real-time recognition tasks using state-of-the-art recognition algorithms exceed the performance level of current embedded processors, and require modern high-performance processors that consume far more power than dedicated hardware [2]–[5]. Dedicated hardware, which is optimized for low-power, real-time recognition tasks, is more suitable for implementing natural human interfaces in low-power mobile embedded systems. Fast and memory-efficient VLSI architectures with small number of registers and processing elements are required for the development of well-optimized embedded systems with capable future human interfaces.

Manuscript received October 20, 2008.
Manuscript revised August 28, 2009.
[†]The authors are with the Graduate School of Information Science, Nagoya University, Nagoya-shi, 464–8603 Japan.
[*]The preliminary version was presented at [1].
a) E-mail: nakamura@is.nagoya-u.ac.jp
DOI: 10.1587/transinf.E93.D.300

VLSI architectures optimized for recognition tasks with low power dissipation have been developed [2]–[6]. Yoshizawa *et al.* investigated a block-wise parallel processing method for output probability computations of continuous hidden Markof models (HMMs) and proposed a low power, high-speed VLSI architecture [2]–[4]. Output probability computations are the most time-consuming part of HMM-based recognition systems. Mathew *et al.* developed low-power accelerators for the SPHINX 3 [7] speech recognition system [5] and perception accelerators for embedded systems [6].

Robust VLSI architecture for the increase of HMM states, which requires small number of registers and processing elements even when the number of HMM states is increased for accurate recognition, is required for the development of well-optimized future HMM-based recognition systems.

In this paper, we present a fast and memory-efficient VLSI architecture for HMM computations using a new blockwise parallel processing method. We show store-based block parallel processing (StoreBPP) for HMM computations, and present an appropriate VLSI architecture for its implementation. Compared with a conventional stream-based block parallel processing (StreamBPP) architecture [2]–[4], when there are a sufficient number of HMM states for accurate recognition, the proposed architecture requires fewer registers and processing elements and less processing time. A comparison demonstrates the efficiency of the proposed architecture through its efficient use of registers in storing input feature vectors and intermediate results for the computations. The processing elements used in the StreamBPP and StoreBPP architectures are identical.

The remainder of this paper is organized as follows: the structure of HMM-based recognition systems is described in Sect. 2, StoreBPP and our VLSI architecture are introduced in Sect. 3, the evaluation of the proposed architecture is described in Sect. 4, and conclusions are presented in Sect. 5.

2. HMM-Based Recognition Systems

2.1 HMM-Based Recognition Hardware

Due to their effectiveness and efficiency for user-independent recognition, HMMs are widely used in appli-

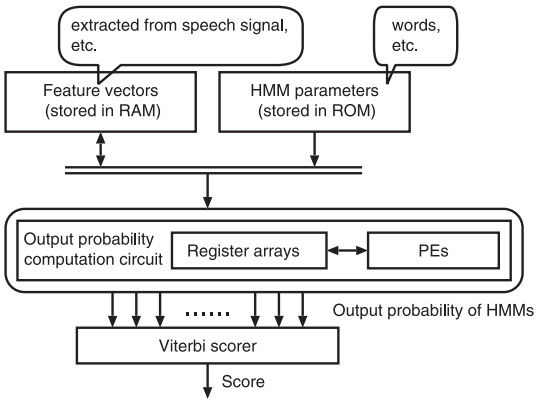

Fig. 1 Basic structure of HMM-based recognition hardware.

cations such as speech recognition, lip-reading, and gesture recognition.

Figure 1 shows the basic structure of HMM-based recognition hardware [2]–[5]. 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 and output probability computation circuit 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 the output probability computation circuit accelerates these computations.

The output probability computation circuit has several register arrays and processing elements (PEs) for efficient high-speed parallel processing.

Typical application examples of our VLSI architecture for the output probability computation circuit are speech recognition systems such as isolated word recognition, connected word recognition and continuous speech recognition, where feature vectors are extracted from input speech signal by any other external circuit or processor outside the recognition hardware of Fig. 1. We assume that the output probability computation circuit computes output probability of continuous HMMs. Model parameters of continuous HMMs are precomputed from training samples of words, etc. In isolated word recognition, the results of the output probability computation and the likelihood score lead to a recognition result, which is a word. In connected word recognition and continuous speech recognition, the results of the output probability computation and the likelihood score are used to lead to a recognition result, which is a sequence of words or a sentence, by any other external circuit or processor outside the recognition hardware of Fig. 1. Our VLSI architecture is directly appliciable to the design of the output probability computation circuit in the systems without a major redesign of the whole hardware of Fig. 1. By utilizing our VLSI architecture to the design of the output probability computation circuit, further design optimization which reduces the number of PEs in the Viterbi scorer by introducing several registers in it is possible but it is our future work.

2.2 Output Probability Computation of HMMs

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

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

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 [2], [3], T, N, P, and V are 86, 32, 38, and 800, respectively, and for another word recognition system [4], T, N, P, and V are 89, 12, 16 and 100 respectively, where the number of HMMs V of the recent system is eight times larger than that of the other system and the number of HMM states of the recent system is increased from 12 to 32 for accurate recognition with the sufficient number of HMM states. For a continuous speech recognition system [5], T, N, P, and V are approximately 20, 10, 40, and 50, respectively. Different applications require different output probability computation circuit architectures.

A flowchart of output probability computations is shown in Fig. 2. 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, a subtraction, an addition, and two multiplications for Eq. (1), and computes $\log b_j(\mathbf{O}_t)$.

3. Fast and Memory-Efficient VLSI Architecture for Output Probability Computations

3.1 Store-Based Block Parallel Processing (StoreBPP)

Block parallel processing (BPP) for output probability computations was proposed as an efficient parallel processing method for word HMM-based speech recognition by Yoshizawa *et al.* [2]–[4]. In this method, the set of input feature 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

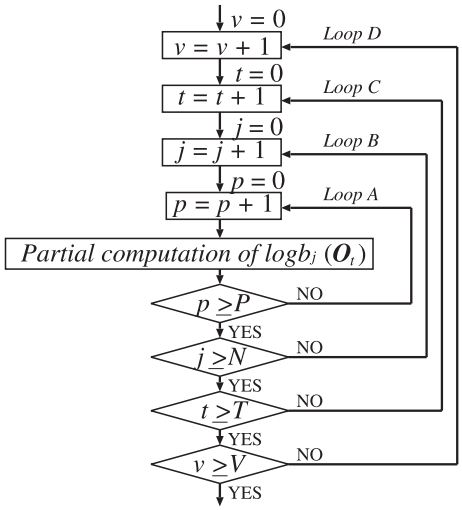

Fig. 2 Flowchart of output probability computation.

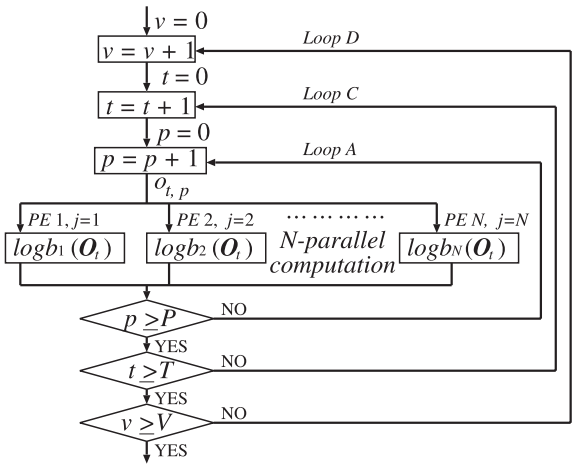

Fig. 3 Flowchart of output probability computations using StreamBPP.

their BPP.

In this paper, we classify two types of BPP according to input data flow: store-based block parallel processing (StoreBPP) and stream-based block parallel processing (StreamBPP). 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 $o_{11}, \ldots, o_{1P}, o_{21}, \ldots, o_{2P}, \ldots, o_{M1}, \ldots, o_{MP}$.

The BPP proposed by Yoshizawa *et al.* [2]–[4] is classified as a StreamBPP. In this paper, we present StoreBPP for output probability computations. $M/2$-parallel computations are performed by our StoreBPP.

A flowchart of the output probability computations with the conventional StreamBPP [2]–[4] is shown in Fig. 3. *PEi* represents the i-th processing element, which computes

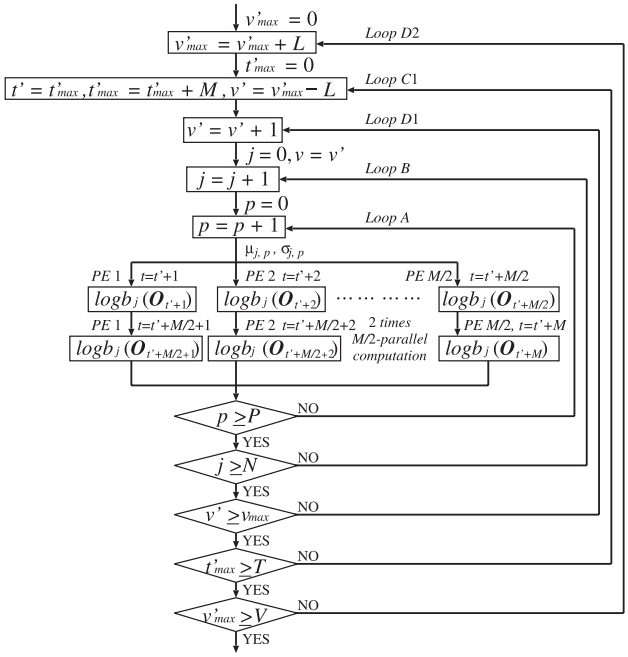

Fig. 4 Flowchart of output probability computations using StoreBPP.

$\log b_j(\mathbf{O}_t)$ by a subtraction, an addition, and two multiplications for Eq. (1). 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 *PEs*. 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 is shown in Fig. 4. The *PEs* 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 $M/2$ *PEs* in Loop C1. In addition to the $M/2$-parallel computations, $\log b_j(\mathbf{O}_{t'+M/2+1})$, $\log b_j(\mathbf{O}_{t'+M/2+2})$, ..., and $\log b_j(\mathbf{O}_{t'+M})$ are also computed with the same $M/2$ *PEs*. In this double $M/2$-parallel computation, the same HMM parameters μ_{jp} and σ_{jp} are used two times, because the parameters are independent of t. In addition to the $M/2$-parallel computations, Loop D (Fig. 2) is divided into Loops D1 and D2 (Fig. 4). The same feature vectors $\mathbf{O}_{t'+1}$, $\mathbf{O}_{t'+2}$, ..., and $\mathbf{O}_{t'+M}$ are used repeatedly during Loop D1, because the input vectors are independent of v.

3.2 A New VLSI Architecture for Output Probability Computation

Our StoreBPP VLSI architecture for output probability computations is shown in Fig. 5. The architecture consists of five register arrays and $M/2$ *PEs*. Reg$\mathbf{O}$ stores M input feature vectors $\mathbf{O}_{t'+1}$, $\mathbf{O}_{t'+2}$, ..., $\mathbf{O}_{t'+M}$. Regμ and Regσ store HMM parameters $-\mu_{jp}$, and σ_{jp}, respectively. Regμ has space for storing $-\mu_{jp}$ and for prestoring $-\mu_{jp+1}$ before the computation with μ_{jp+1} during the computation using

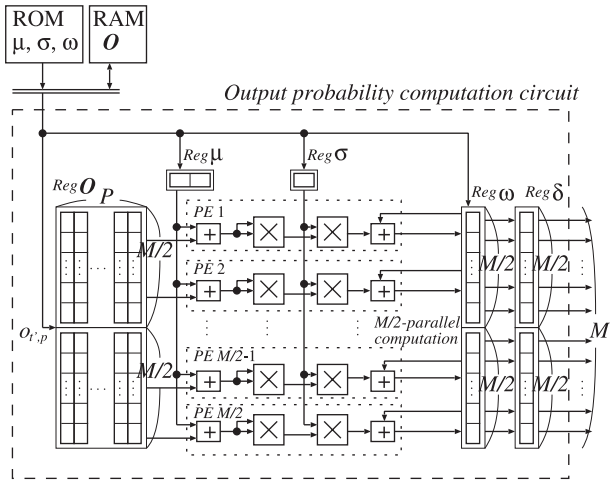

Fig. 5 StoreBPP VLSI architecture.

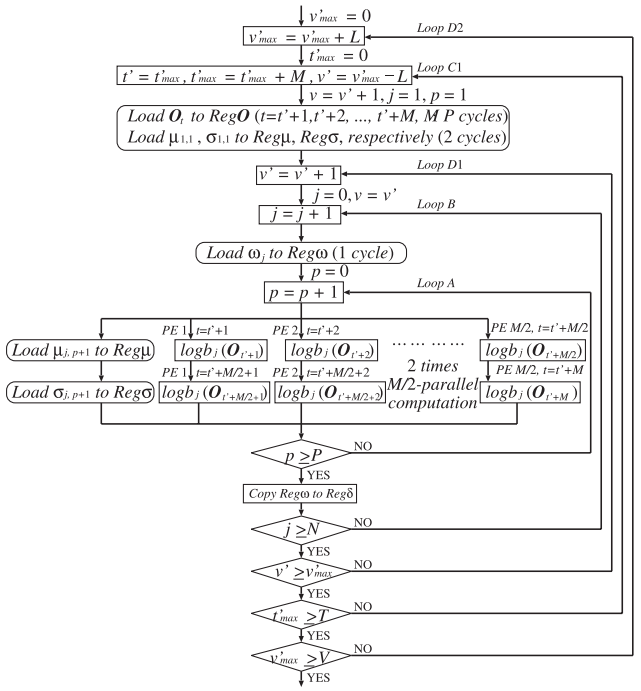

Fig. 6 Flowchart of computations using the StoreBPP architecture.

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 *PE* consists of two adders and two multipliers, which are used for computing $\omega_j + \sum_p^P \sigma_{jp}(o_{tp} - \mu_{jp})^2$.

Figure 6 shows the flowchart of output probability computations using the StoreBPP architecture. The computation starts by reading M input feature vectors from RAM and storing them to Reg$\mathbf{O}$ in Loop C1. The HMM parameters of v-th HMM are read from ROM and stored in Regμ, Regσ, and Regω, which are μ_{11}, σ_{11}, and ω_1. For the first half of the stored input feature vectors $\mathbf{O}_{t'+1}$, $\mathbf{O}_{t'+2}$, ..., and $\mathbf{O}_{t'+M/2}$, $M/2$ intermediate results are simultaneously computed with the stored μ_{11} and σ_{11} by $M/2$ *PEs*, where the HMM parameters are shared by all *PEs*. At the same time, an HMM parameter μ_{jp+1} of v-th HMM is read from ROM and stored in Regμ, where Regμ still holds μ_{11} for the next computation using μ_{11}. Then, for the other half of the stored input feature vectors $\mathbf{O}_{t'+M/2+1}$, $\mathbf{O}_{t'+M/2+2}$, ..., and $\mathbf{O}_{t'+M}$, $M/2$ intermediate results are simultaneously computed with the same μ_{11} and σ_{11} by $M/2$ *PEs*. At the same time, an HMM parameter σ_{jp+1} of v-th HMM is read from ROM and stored in Regσ, where the value is overwritten because the computation with σ_{11} has been finished. In this double $M/2$-parallel computation, the stored HMM parameters μ_{11} and σ_{11} are used two times. In the next double $M/2$-parallel computation, the stored HMM parameters μ_{jp+1} and σ_{jp+1} are used two times. M output probabilities $\log b_j(\mathbf{O}_{t'+1})$, $\log b_j(\mathbf{O}_{t'+2})$, ..., and $\log b_j(\mathbf{O}_{t'+M})$ of v-th HMM are obtained by Loop A. The obtained results are copied from Regω to Regδ for starting the next output probability computation, $\log b_{j+1}(\mathbf{O}_{t'+1})$, $\log b_{j+1}(\mathbf{O}_{t'+2})$, ..., $\log b_{j+1}(\mathbf{O}_{t'+M})$. The stored results are fed to the Viterbi scorer. The $M \cdot N$ output probabilities of v-th HMM are obtained by Loop B. $M \cdot N \cdot L$ output probabilities of HMM v', $v'+1$, ..., $v'+L-1$ are obtained by Loop D1 with the same M input feature vectors $\mathbf{O}_{t'+1}$, $\mathbf{O}_{t'+2}$, ..., $\mathbf{O}_{t'+M}$. The $M \cdot N \cdot L \cdot T/M$ output probabilities of HMM v', $v'+1$, ..., $v'+L-1$ are obtained by Loop C1, and finally the $M \cdot N \cdot L \cdot T/M \cdot V/L$ output probabilities of all HMMs are obtained by Loop D2.

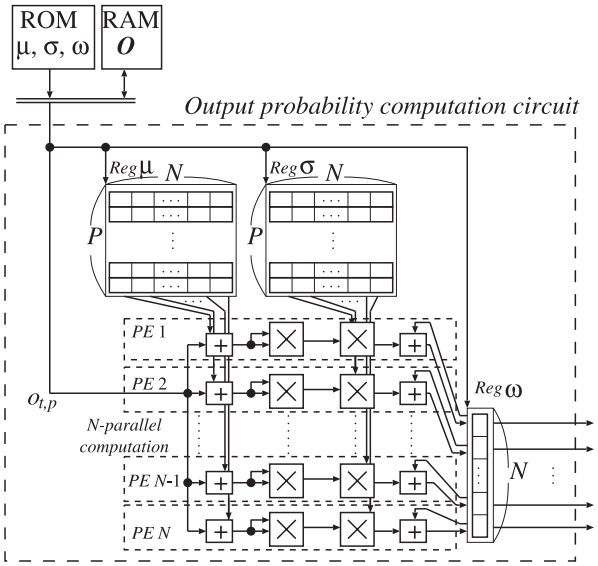

Fig. 7 StreamBPP VLSI architecture.

4. Evaluation

We compared the proposed StoreBPP and StreamBPP (Fig. 7) VLSI architecture [2]–[4]. The StreamBPP architecture consists of three register arrays and N *PEs*. Regμ and Regσ store HMM parameters $-\mu_{jp}$ and σ_{jp}, respectively, and Regω stores HMM parameter ω_j and intermediate re-

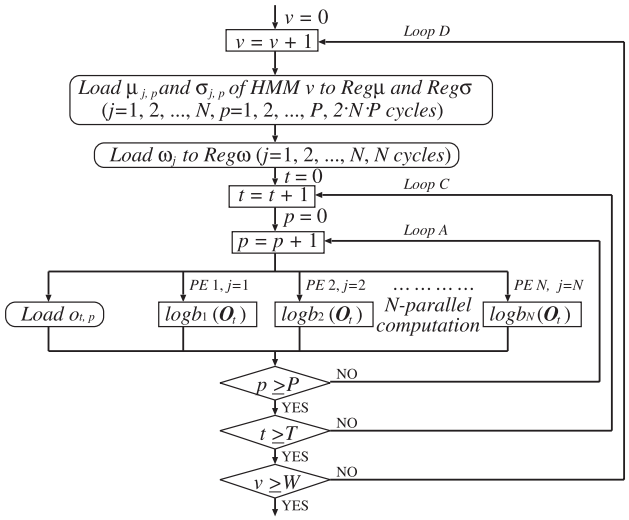

Fig. 8 Flowchart of computations using StreamBPP architecture.

Table 1 Register size.

	Register size (bit)
StoreBPP (ours)	$P \cdot M \cdot x_o + 2 \cdot x_\mu + x_\sigma + 2 \cdot M \cdot x_f$
StreamBPP	$N \cdot P \cdot x_\mu + N \cdot P \cdot x_\sigma + N \cdot x_f$

Table 2 Processing times.

	Processing time (cycles)
StoreBPP (ours)	$\lceil V/L \rceil$ $\cdot \{P \cdot M + (1 + 2 \cdot P) \cdot L \cdot N\} \cdot \lceil T/M \rceil$
StreamBPP	$V \cdot (2 \cdot N \cdot P + N + P \cdot T)$

sults. The *PEs* in Figs. 7 and 5 are identical.

Figure 8 shows the flowchart of the computations of the StreamBPP architecture. The computation starts by reading all $2 \cdot N \cdot P + N$ HMM parameters of v-th HMM from ROM and storing them to Regμ, Regσ, and Regω in Loop D. For stream input o_{tp}, the intermediate results are computed with stored HMM parameters by N *PEs*. N output probabilities $\log b_1(\mathbf{O}_t)$, $\log b_2(\mathbf{O}_t)$, ..., $\log b_N(\mathbf{O}_t)$ of the HMM are obtained by Loop A. The obtained results are fed to a Viterbi scorer. $N \cdot T$ output probabilities of v-th HMM are obtained by Loop C with the same HMM parameters. The $N \cdot T \cdot V$ output probabilities of all HMMs are obtained by Loop D.

Table 1 shows the register size of the StoreBPP and StreamBPP architectures, where x_μ, x_σ, x_o, and x_f represent the bit length of μ_{jp}, σ_{jp}, o_{tp}, and the output of *PE*, 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 with the 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 Fig. 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 $N = 32$, $P = 38$, $T = 86$, $x_\mu = 8$, $x_\sigma = 8$, $x_f = 24$, $x_o = 8$, and $V = 800$—the same values used in a recent circuit design for isolated word recognition [2], [3]. We also assume that $M = 44$ and $L = 5$ for the StoreBPP architecture. The *PEs* used in the StreamBPP and StoreBPP architectures are identical. Compared with the StreamBPP architecture, the StoreBPP architecture has fewer registers, requires less processing time, and has fewer *PEs*. From the VLSI architectural viewpoint, this is because the register size and the number of *PEs* of the StoreBPP architecture are independent of N, and its *PEs* can repeatedly use the same input feature vectors. The StoreBPP architecture has fewer wait cycles for reading data from ROM before parallel computations—586,240 ($\lceil V/L \rceil \cdot (P \cdot M + L \cdot N) \cdot \lceil T/M \rceil$)—than the StreamBPP system, which has 1,971,200 ($V \cdot \{2 \cdot N \cdot P + N\}$).

Table 3 Evaluation of the StreamBPP and StoreBPP performance.

	Register size (bit)	Processing time (cycles)	#*PEs*
StoreBPP (ours)	15,512	4,477,440	22
StreamBPP	20,224	4,585,600	32

From a logic design viewpoint, the register arrays of the StreamBPP and StoreBPP 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. 8 and 6. The data path delay is the same for both the StreamBPP and StoreBPP designs—equal to the delay time of one *PE*. The delay times of control paths differ between the two, but the control path delay is small compared with the data path delay.

5. Conclusions

We presented StoreBPP for output probability computations and presented a new VLSI architecture. StoreBPP performs arithmetic operations to locally stored input feature vectors. Compared with the conventional StreamBPP architecture, when the number of HMM states is large enough for accurate recognition, the StoreBPP architecture requires fewer registers and *PEs*, and less processing time. In terms of the VLSI architecture, a comparison shows the efficiency of the proposed architecture. A logic design, a Viterbi scorer for the StoreBPP architecture, and a reconfigurable architecture for both the StreamBPP and StoreBPP architectures are our future works.

Acknowledgments

We would like to express our thanks to all members of Takagi Laboratory at Nagoya University for their discussions and comments. Thanks are due to reviewers and the editor for their helpful comments. We also thank to Assistant Professor Shingo Yoshizawa and Professor Yoshikazu

Miyanaga of Hokkaido University for their helpful comments.

References

[1] K. Nakamura, M. Yamamoto, K. Takagi, and N. Takagi, "Fast and memory efficient VLSI architecture for output probability computations of HMM-based recognition systems," Proc. 2008 IEEE Int'l Symposium on Circuits and Systems (ISCAS'08), pp.1688–1691, 2008.
[2] S. Yoshizawa, N. Wada, N. Hayasaka, and Y. Miyanaga, "Scalable architecture for word HMM-based speech recognition and VLSI implementation in complete system," IEEE Trans. Circuits Syst., vol.53, no.1, pp.70–77, 2006.
[3] S. Yoshizawa, N. Wada, N. Hayasaka, and Y. Miyanaga, "Scalable architecture for word HMM-based speech recognition," Proc. 2004 IEEE Int'l Symposium on Circuits and Systems (ISCAS'04), pp.417–420, 2004.
[4] S. Yoshizawa, Y. Miyanaga, and N. Yoshida, "On a high-speed HMM VLSI module with block parallel processing," IEICE Trans. Fundamentals (Japanese Edition), vol.J85-A, no.12, pp.1440–1450, Dec. 2002.
[5] B. Mathew, A. Davis, and Z. Fang, "A low-power accelerator for the SPHINX 3 speech recognition system," Proc. Int'l Conf. on Compilers, Architecture and Synthesis for Embedded Systems, pp.210–219, 2003.
[6] B. Mathew, A. Davis, and A. Ibrahim, "Perception coprocessors for embedded systems," Proc. Workshop on Embedded Systems for Real-Time Multimedia (ESTIMedia), pp.109–116, 2003.
[7] X. Huang, F. Alleva, H.W. Hon, M.Y. Hwang, K.F. Lee, and R. Rosenfeld, "The SPHINX-II speech recognition system: An overview," Comput. Speech Lang., vol.7, no.2, pp.137–148, 1992.

Kazuhiro Nakamura received the B.S. degree in information science from Shimane University in 1997. He received the M.E. and Dr.Eng. degrees in information science from Nara Institute of Science and Technology, Nara, Japan, in 1998, and 2001, respectively. Since 2001, he has been a Research Associate at Nagoya University. His current interests include VLSI CAD algorithms, hardware algorithms and VLSI architecture.

Masatoshi Yamamoto received the B.E. and M.I.S. degrees in information engineering from Nagoya University, Nagoya, Japan, in 2005 and 2008, respectively. His current interests include hardware algorithms and VLSI architecture.

Kazuyoshi Takagi received the B.E., M.E. and Dr. of Engineering degrees in information science from Kyoto University, Kyoto, Japan, in 1991, 1993 and 1999 respectively. From 1995 to 1999, he was a Research Associate at Nara Institute of Science and Technology. He has been an Assistant Professor since 1999 and promoted to an Associate Professor in 2006, at the Department of Information Engineering, Nagoya University, Nagoya, Japan. His current interests include circuit complexity theory, VLSI design, and VLSI CAD algorithms.

Naofumi Takagi received the B.E., M.E., and Ph.D. degrees in information science from Kyoto University, Kyoto, Japan, in 1981, 1983, and 1988, respectively. He joined Department of Information Science, Kyoto University, as an instructor in 1984 and was promoted to an associate professor in 1991. He moved to Department of Information Engineering, Nagoya University, Nagoya, Japan, in 1994, where he has been a professor since 1998. His current interests include computer arithmetic, hardware algorithms, and logic design. He received Japan IBM Science Award and Sakai Memorial Award of the Information Processing Society of Japan in 1995.