We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

6,300

Open access books available

172,000

International authors and editors

190M

Downloads

Our authors are among the

154

Countries delivered to

TOP 1%

most cited scientists

12.2%

Contributors from top 500 universities

WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index
in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com

A VLSI Architecture for Output Probability Computations of HMM-based Recognition Systems

Kazuhiro Nakamura, Masatoshi Yamamoto,
Kazuyoshi Takagi and Naofumi Takagi
Nagoya University
Japan

1. Introduction

Mobile embedded systems with 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. 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. VLSI architectures optimized for recognition tasks with low power dissipation have been developed. Yoshizawa et al. investigated a block-wise parallel processing method for output probability computations of continuous hidden Markov models (HMMs), and proposed a low power, high-speed VLSI architecture. 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 speech recognition system, and also developed perception accelerators for embedded systems. In this chapter, we present a fast and memory efficient VLSI architecture for output probability computations of continuous HMMs using a new block-wise parallel processing method. We show block-wise frame parallel processing (BFPP) for output probability computations and present an appropriate VLSI architecture for its implementation. Compared with a conventional block-wise state parallel processing (BSPP) architecture, when there are a sufficient number of HMM states for accurate recognition, the BFPP architecture requires fewer registers and processing elements (*PEs*), and less processing time. The *PEs* used in the BFPP architecture are identical to those used in the BSPP architecture. From a VLSI architectural viewpoint, a comparison shows the efficiency of the BFPP architecture through efficient use of registers for storing input feature vectors and intermediate results during computation. The remainder of this chapter is organized as follows: the structure of HMM based recognition systems is described in

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

Section 2, BFPP and BFPP-based VLSI architecture are introduced in Section 3, the evaluation of the BFPP architecture is described in Section 4, and conclusions are presented in Section 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 applications such as speech recognition, lip-reading, and gesture recognition. Figure 1 shows the basic structure of HMM-based recognition hardware (Yoshizawa et al., 2006, Yoshizawa et al., 2004, Yoshizawa et al., 2002, Mathew et al., 2003a). 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.

2.2 Output Probability Computation of HMMs

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

Fig. 2. Flowchart of output probability computation

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

where ω_j , σ_{jp} and μ_{jp} are the factors of the Gaussian probability density function (Yoshizawa et al., 2006).

The output probability computation circuit 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 frames 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 (Yoshizawa et al., 2006, Yoshizawa et al., 2004), T , N , P and V are 86, 32, 38, and 800, respectively, and for another word recognition system (Yoshizawa et al., 2002), T , N , P and V are 89, 12, 16 and 100 respectively. For a continuous speech recognition system (Mathew et al., 2003a), 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 for V HMMs is shown in Fig. 2. Output probabilities are obtained by $T \times N \times P \times V$ times the partial computation of $\log b_j(\mathbf{O}_t)$. Partial computation of $\log b_j(\mathbf{O}_t)$ performs four arithmetic operations, a subtraction ($a = o_{tp} - \mu_{jp}$), an addition ($acc = acc + b$, where the initial value of acc is ω_j), and two multiplications ($b = a \times a \times \sigma_{jp}$) for Eq. (1), and computes $\log b_j(\mathbf{O}_t)$.

3. Fast and memory efficient VLSI architecture

3.1 Block-wise frame parallel processing

Block parallel processing (BPP) for output probability computations was proposed as an efficient parallel processing method for word HMM-based speech recognition by Yoshizawa

Fig. 3. Flowchart of output probability computation using BSPP

et al. (Yoshizawa et al., 2006, Yoshizawa et al., 2004, Yoshizawa et al., 2002). In this method, the set of input frames is called a *block*, and HMM parameters are effectively shared between different input frames in the computation. N -parallel computation is performed by their BPP. In this chapter, we classify two types of BPP according to data flow of output probability computations: block-wise frame parallel processing (BFPP) and block-wise state parallel processing (BSPP). A block can be seen as a set of M ($\leq T$) input frames, whose elements are $\mathbf{O}_{t'}$'s, $1 \leq t' \leq M$. M frames in T input frames are processed in block. *BFPP* performs arithmetic operations to locally stored input frames, which are $\mathbf{O}_1, \mathbf{O}_2, \dots, \mathbf{O}_M$, and output probability computations for multiple frames are carried out simultaneously. On the other hand, a block can also be seen as a $M \times P$ matrix whose elements are $o_{t'p}$, $1 \leq t' \leq M, 1 \leq p \leq P$. *BSPP* performs arithmetic operations to an input sequence, which is $o_{11}, \dots, o_{1P}, o_{21}, \dots, o_{2P}, \dots, o_{M1}, \dots, o_{MP}$, and output probability computations for multiple states are carried out simultaneously.

The BPP proposed by Yoshizawa et al. (Yoshizawa et al., 2006, Yoshizawa et al., 2004, Yoshizawa et al., 2002) is classified as a BSPP. In this chapter, we present BFPP for output probability computations. $M/2$ -parallel computations are performed by our BFPP.

A flowchart of the output probability computations with the conventional BSPP (Yoshizawa et al., 2006, Yoshizawa et al., 2004, Yoshizawa et al., 2002) is shown in Fig. 3.

PE_i represents the i -th processing element, which computes $\log b_i(\mathbf{O}_t)$ by a subtraction, an addition, and two multiplications for Eq. (1). *Loop B* (Fig. 2) is expanded as shown in Fig. 3, and $\log b_1(\mathbf{O}_t), \log b_2(\mathbf{O}_t), \dots$, and $\log b_N(\mathbf{O}_t)$ are computed simultaneously with N PEs, where o_{tp} is fed to the N PEs in *Loop A*. In addition to the N -state parallel computation, the same HMM parameters μ_{jp}' 's and 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 BFPP is shown in Fig. 4. The PEs in

Fig. 4. Flowchart of output probability computation using BFPP

Figs. 4 and 3 are identical, but in a different number. *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*, where μ_{jp} and σ_{jp} are fed to the $M/2$ PEs in *Loop A*. In addition to the $M/2$ -frame 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 twice, 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 input frames $\mathbf{O}_{t'+1}$, $\mathbf{O}_{t'+2}$, ..., and $\mathbf{O}_{t'+M}$ are used repeatedly during *Loop D1*, because the input frames are independent of v .

3.2 A VLSI architecture for output probability computation

Our BFPP VLSI architecture for output probability computations is shown in Fig. 5. The architecture consists of five register arrays and $M/2$ PEs. *RegO* stores M input frames $O_{t'+1}$, $O_{t'+2}$, ..., $O_{t'+M}$. *Regμ* and *Regσ* store HMM parameters $-\mu_{jp}$, and σ_{jp} , respectively. *Regω* stores

Fig. 5. BFPP VLSI architecture

HMM parameter ω_j and intermediate results. *Regδ* stores computed output probabilities for a Viterbi scorer. Each PE_i consists of two adders and two multipliers, which are used for computing $\omega_j + \sum_{p=1}^P \sigma_{jp} (o_{t'p} - \mu_{jp})^2$.

Figure 6 shows the flowchart of output probability computations using the BFPP architecture. The computation starts by reading M input frames from RAM and storing them to *RegO* in Loop C1, which are $O_{t'+1}$, $O_{t'+2}$, ..., $O_{t'+M/2}$, $O_{t'+M/2+1}$, $O_{t'+M/2+2}$, ..., $O_{t'+M}$. 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 . The value of all registers in *Regω* is set to ω_1 . For the first half of the stored input frames $O_{t'+1}$, $O_{t'+2}$, ..., and $O_{t'+M/2}$, $M/2$ intermediate results are simultaneously computed with the stored μ_{11} , σ_{11} , and ω_1 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 $Reg\mu$. Then, for the other half of the stored input frames $O_{t'+M/2+1}$, $O_{t'+M/2+2}$, ..., and $O_{t'+M}$, $M/2$ intermediate results are simultaneously computed with the same μ_{11} , σ_{11} , and ω_1 by $M/2$ PEs. At the same time, an HMM parameter $\sigma_{j\ p+1}$ of v -th HMM is read from ROM and stored in $Reg\sigma$. In this double $M/2$ -parallel computation, the

Fig. 6. Flowchart of computations using the BFPP architecture

same HMM parameters μ_{11} , σ_{11} , and ω_1 are used twice. In the next double $M/2$ -parallel computation, the stored HMM parameters $\mu_{j\ p+1}$ and $\sigma_{j\ p+1}$ are used twice. M output probabilities $\log b_j(O_{t'+1})$, $\log b_j(O_{t'+2})$, ..., and $\log b_j(O_{t'+M})$ of v -th HMM are obtained by *Loop*

A. The obtained results are transferred from $Reg\omega$ to $Reg\delta$ for starting the next output probability computation, $\log b_{j+1}(\mathbf{O}_{t'+1})$, $\log b_{j+1}(\mathbf{O}_{t'+2})$, ..., $\log b_{j+1}(\mathbf{O}_{t'+M})$ of v -th HMM. 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 $v' + 1$, $v' + 2$, ..., $v' + L$ are obtained by *Loop D1* with the same M input frames $\mathbf{O}_{t'+1}$, $\mathbf{O}_{t'+2}$, ..., and frames $\mathbf{O}_{t'+1}$, $\mathbf{O}_{t'+2}$, ..., and $\mathbf{O}_{t'+M}$.

Fig. 7. BSPP VLSI architecture

frames $\mathbf{O}_{t'+1}$, $\mathbf{O}_{t'+2}$, ..., and $\mathbf{O}_{t'+M}$. The $M \cdot N \cdot L \cdot (T/M)$ output probabilities of HMM $v' + 1$, $v' + 2$, ..., $v' + L$ 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*.

4. Evaluation

We compared the proposed BFPP with BSPP (Fig. 7) VLSI architecture (Yoshizawa et al., 2006, Yoshizawa et al., 2004, Yoshizawa et al., 2002). The BSPP architecture consists of three register arrays and N PEs. $Reg\mu$ and $Reg\sigma$ store HMM parameters $-\mu_{jp}$ and σ_{jp} , respectively, and $Reg\omega$ stores HMM parameter ω_j and intermediate results. The PEs in Figs. 7 and 5 are identical.

Figure 8 shows the flowchart of the computations of BSPP 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 $Reg\mu$, $Reg\sigma$, and $Reg\omega$ in *Loop D*. For 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. $N \cdot T$

Fig. 8. Flowchart of computations using the BSPP architecture

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

Table 1. Register size

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

Table 2. Processing time

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 BSPP and BFPP 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 (frame), and the number of input frames in a block, respectively.

Table 2 shows the processing time for computing output probabilities of V HMMs with the BFPP and BSPP architectures, where T and L are the number of input frames and the number of HMMs whose output probabilities are computed with the same input frames during *Loop D1* of Fig. 6, respectively.

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

Table 3. Evaluation of the BSPP and BFPP performance

Fig. 9. Evaluation of the BSPP and BFPP performance, and the value of M of the BFPP ($N = 32$, $P = 38$, $T = 86$, $V = 800$)

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 $N = 32$, $P = 38$, $T = 86$, $x_{\mu} = 8$, $x_{\sigma} = 8$, $x_o = 8$, $x_f = 24$, and $V = 800$, the same values used in a recent circuit design for isolated word recognition (Yoshizawa et al., 2006, Yoshizawa et al., 2004). We also assume that $M = 44$ and $L = 5$ for the BFPP architecture. The PEs used in the BSPP and BFPP architectures are identical. Compared with the BSPP architecture, the BFPP architecture has fewer registers, requires less processing time, and has fewer PEs . From the VLSI architecture viewpoint, this is because the register size of the BFPP architecture is independent of N , and its PEs can repeatedly use the same input frames. The BFPP 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 BSPP architecture, which has $1,971,200 (V \cdot (2 \cdot N \cdot P + N))$.

Fig. 9 shows the processing time and the number of *PEs* of the BFPP and BSPP architectures, and the value of *M* of the BFPP architecture. The processing time and the number of *PEs* of the BFPP architecture are less than those of the BSPP architecture when $M = 44$ (Fig. 9).

From a logic design viewpoint, the register arrays of the BSPP and BFPP 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 BSPP and BFPP 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 BFPP for output probability computations and presented an appropriate VLSI architecture for its implementation. BFPP performs arithmetic operations to locally stored input frames, and output probability computations for multiple frames are carried out simultaneously. Compared with the conventional BSPP architecture, when the number of HMM states is large enough for accurate recognition, the BFPP architecture requires fewer registers and *PEs*, and less processing time. In terms of the VLSI architecture, a fast and memory efficient VLSI architecture for output probability computations of HMM-based recognition systems has been presented. A logic design, a Viterbi scorer for the BFPP architecture, and a reconfigurable architecture for both the BSPP and BFPP architectures are our future works.

6. References

- B. Mathew, A. Davis & Z. Fang (2003a). Perception Coprocessors for Embedded Systems, *Proc. of Workshop on Embedded Systems for Real-Time Multimedia (ESTIMedia)*, pp. 109-116, 2003.
- B. Mathew, A. Davis & Z. Fang (2003b). 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.
- S. Yoshizawa, Y. Miyanaga & N. Yoshida (2002). On a High-Speed HMM VLSI Module with Block Parallel Processing, *IEICE Trans. Fundamentals (Japanese Edition)*, Vol. J85-A, No. 12, pp. 1440-1450, 2002.
- S. Yoshizawa, N. Wada, N. Hayasaka & Y. Miyanaga (2004). Scalable Architecture for Word HMM-Based Speech Recognition, *Proc. of 2004 IEEE Int'l Symposium on Circuits and Systems (ISCAS'04)*, pp. 417-420, 2004.
- S. Yoshizawa, N. Wada, N. Hayasaka & Y. Miyanaga (2006). Scalable Architecture for Word HMM-Based Speech Recognition and VLSI Implementation in Complete System, *IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS*, Vol. 53, No. 1, pp. 70-77, 2006.

- X. Huang, F. Alleva, H. W. Hon, M. Y. Hwang, K. f. Lee & R. Rosenfeld (1992). The SPHINX-II speech recognition system: an overview, *Computer Speech and Language*, Vol. 7(2), pp. 137-148, 1992.

IntechOpen

IntechOpen

VLSI

Edited by Zhongfeng Wang

ISBN 978-953-307-049-0

Hard cover, 456 pages

Publisher InTech

Published online 01, February, 2010

Published in print edition February, 2010

The process of Integrated Circuits (IC) started its era of VLSI (Very Large Scale Integration) in 1970's when thousands of transistors were integrated into one single chip. Nowadays we are able to integrate more than a billion transistors on a single chip. However, the term "VLSI" is still being used, though there was some effort to coin a new term ULSI (Ultra-Large Scale Integration) for fine distinctions many years ago. VLSI technology has brought tremendous benefits to our everyday life since its occurrence. VLSI circuits are used everywhere, real applications include microprocessors in a personal computer or workstation, chips in a graphic card, digital camera or camcorder, chips in a cell phone or a portable computing device, and embedded processors in an automobile, et al. VLSI covers many phases of design and fabrication of integrated circuits. For a commercial chip design, it involves system definition, VLSI architecture design and optimization, RTL (register transfer language) coding, (pre- and post-synthesis) simulation and verification, synthesis, place and route, timing analyses and timing closure, and multi-step semiconductor device fabrication including wafer processing, die preparation, IC packaging and testing, et al. As the process technology scales down, hundreds or even thousands of millions of transistors are integrated into one single chip. Hence, more and more complicated systems can be integrated into a single chip, the so-called System-on-chip (SoC), which brings to VLSI engineers ever increasingly challenges to master techniques in various phases of VLSI design. For modern SoC design, practical applications are usually speed hungry. For instance, Ethernet standard has evolved from 10Mbps to 10Gbps. Now the specification for 100Mbps Ethernet is on the way. On the other hand, with the popularity of wireless and portable computing devices, low power consumption has become extremely critical. To meet these contradicting requirements, VLSI designers have to perform optimizations at all levels of design. This book is intended to cover a wide range of VLSI design topics. The book can be roughly partitioned into four parts. Part I is mainly focused on algorithmic level and architectural level VLSI design and optimization for image and video signal processing systems. Part II addresses VLSI design optimizations for cryptography and error correction coding. Part III discusses general SoC design techniques as well as other application-specific VLSI design optimizations. The last part will cover generic nano-scale circuit-level design techniques.

How to reference

In order to correctly reference this scholarly work, feel free to copy and paste the following:

Kazuhiro Nakamura, Masatoshi Yamamoto, Kazuyoshi Takagi and Naofumi Takagi (2010). A VLSI Architecture for Output Probability Computations of HMM-based Recognition Systems, VLSI, Zhongfeng Wang (Ed.), ISBN: 978-953-307-049-0, InTech, Available from: <http://www.intechopen.com/books/vlsi/a-vlsi-architecture-for-output-probability-computations-of-hmm-based-recognition-systems>

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

IntechOpen

IntechOpen

© 2010 The Author(s). Licensee IntechOpen. This chapter is distributed under the terms of the [Creative Commons Attribution-NonCommercial-ShareAlike-3.0 License](https://creativecommons.org/licenses/by-nc-sa/3.0/), which permits use, distribution and reproduction for non-commercial purposes, provided the original is properly cited and derivative works building on this content are distributed under the same license.

IntechOpen

IntechOpen