Parallel System Design for Time-Delay Neural Networks

David Zhang, *Senior Member, IEEE,* and Sankar K. Pal, *Fellow, IEEE*

***Abstract*—In this paper, we develop a parallel structure for the time-delay neural network used in some speech recognition applications. The effectiveness of the design is illustrated by 1) extracting a window computing model from the time-delay neural systems; 2) building its pipelined architecture with parallel or serial processing stages; and 3) applying this parallel window computing to some typical speech recognition systems. An analysis of the complexity of the proposed design shows a greatly reduced complexity while maintaining a high throughput rate.**

***Index Terms*—Parallel computing, pipelined architecture, time-delay neural networks, speech recognition.**

I. Introduction

Artificial neural networks (ANN), as processors of time-sequence patterns, have been successfully applied to several speaker-dependent speech recognition problems [1]–[14]. A variety of neural speech recognition algorithms has been developed. Numerous studies have demonstrated the effectiveness of multilayer systems with time-delay sequences as inputs to these systems [15]–[18]. Typical examples are: time-delay neural network (TDNN) proposed by Waibel and Lang [19]–[21]; block-windowed neural network (BWNN) by Sawai [22]; and dynamic programming neural network (DNN) by Sakoe [23], [24].

Some features used in these neural speech recognition systems are incorporation of time delays, temporal integration, or recurrent connections. Spectral inputs are applied to input nodes sequentially, one frame at a time, and their corresponding input matrix is formed [15], [16]. Since only short time delays are used, these neural speech recognition systems can be integrated into real time speech recognizer. However, these systems concern, so far, mainly with algorithms; their behaviors and characteristics are primarily investigated by simulation on general purpose computers. The spatiotemporal computing parallelism inhered in such neural speech recognition systems is little explored; thereby restricting its application domain to real life problems.

In this paper, we describe a methodology for parallel time-delay window computing by considering the features and characteristics of such neural speech recognition systems. A model for time-delay window computing and its corresponding architecture definition are described in Section II. Two kinds of processing stages used in pipelined architecture and their building elements are explained in Section III. In Section IV, some mapping strategies from window computing model into systolic array structures are defined. Three typical speech recognition applications and their performance analysis by parallel window computing are given in Sections V and VI, respectively. A brief conclusion is included in Section VII.

Manuscript received June 21, 1999; revised February 1, 2000. The work was supported in part by the UGC, Hong Kong, and the Central Fund, Hong Kong Polytechnic University.

The authors are with the Department of Computing, Hong Kong Polytechnic University, Kowloon, Hong Kong. S. K. Pal is on leave from the Machine Intelligence Unit, Indian Statistical Institute, Calcutta 700035, India.

Publisher Item Identifier S 1094-6977(00)04798-2.

II. Window Computing Model

A. Definition and Notation

Based on the neural systems with time-delay sequence input of feature parameters for speech recognition [15]–[18], we can develop a typical computing model composed of $p+2$ layers, which includes an input layer, p hidden layers, and an output layer. Both the input layer and the hidden layers are characterized by time-delay sequence input matrix of speech parameters, built by $m_s \times n_s$ $(s = 1, 2, \cdots, p+1)$ memory elements, where $m_{p+1} = q$ is the number of pattern classes. The output layer consists in q units. The relation between node x_{ij} in Layer s and node y_{kl} in Layer $s+1$ can be defined as

$$y_{kl} = f\left(\sum \sum x_0 w^{kl}_{i-k+1,\, j-i+1} + \theta^{(kl)}\right) \tag{1}$$

where $k \leq i \leq k + e_s - 1$ and $l \leq j \leq l + r_s - 1$; f is a sigmoid function; $w^{(kl)}$ and $\theta^{(kl)}$, $1 \leq k \leq m_{s+1}$ and $1 \leq l \leq n_{s+1}$, are referred to as weight value and bias value, respectively. Both values can be obtained from a small input submatrix (called "window"), where the size is $e_s \times r_s$ $(e_s \leq m_s$ and $r_s \leq n_s)$ in Layer s, to the node y_{kl} in Layer $s+1$. This kind of time-delay window computing methodology is shown in Fig. 1. Obviously, there will be $m_{s+1} \times n_{s+1}$ windows formed by the input matrix $(m_s \times n_s)$ in Layer s.

To implement such a time-delay window computing in (1), we can use only an input window built by $e_s \times r_s$ elements in Layer s. Instead of moving such a window to the whole input matrix, speech parameters in time-delay sequence are arranged to pass through the window in pipeline. Thus, the expression in (1) can be rewritten as

$$y_{kl} = f\left(\sum W_{kl} X^T\right) = f\left(\sum X W^T_{kl}\right) \tag{2}$$

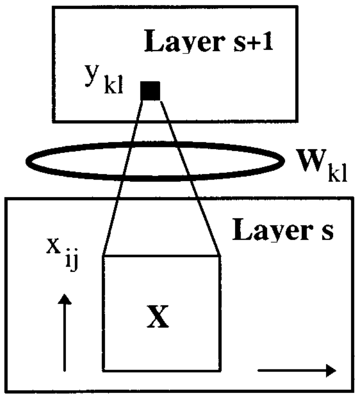


Fig. 1. Time-delay window computing model between layer s and layer $s+1$.

where X is an input sub-matrix given by such a fixed window, which can be represented as

$$X = \begin{bmatrix} X_1 \\ X_2 \\ \cdots \\ X_{e_s} \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1r_s} \\ x_{21} & x_{22} & \cdots & x_{2r_s} \\ \cdots & \cdots & \cdots & \cdots \\ x_{e_s1} & x_{e_s2} & \cdots & x_{e_sr_s} \end{bmatrix} \tag{3}$$

and W_{kl} is the corresponding weight matrix from the window in Layer s to node y_{kl} in Layer $s+1$, i.e.,

$$W_{kl} = \begin{bmatrix} w_{11}^{(kl)} & w_{12}^{(kl)} & \cdots & w_{1r_s}^{(kl)} \\ w_{21}^{(kl)} & w_{22}^{(kl)} & \cdots & w_{2r_s}^{(kl)} \\ \cdots & \cdots & \cdots & \cdots \\ w_{e_s1}^{(kl)} & w_{e_s2}^{(kl)} & \cdots & w_{e_sr_s}^{(kl)} \end{bmatrix}. \tag{4}$$

It is evident that there are $m_{s+1} \times n_{s+1}$ different weight matrices from Layer s to Layer $s+1$. Their sizes are equal to the size of the window in Layer s.

B. Pipelined Neural Architecture

The time-delay window computing model discussed above can be implemented by a pipelined neural architecture with $p+1$ processing stages, each with its own control sequence [see Fig. 2(a)]. In each processing stage, a fixed time-delay computing window is built as a connection to next stage. Loading an input submatrix, X, to the window in a pipeline mode and mapping the corresponding weight matrix, W_{kl} $(k = 1, 2, \cdots, m_{s+1};\ l = 1, 2, \cdots, n_{s+1})$, the output result, y_{kl}, can be obtained. Since all time-delay computing windows in the pipelined neural architecture are capable of working at the same time, the potential parallelism inhered in such neural speech recognition systems can be well explored.

A basic time-delay neuron in the pipelined neural architecture is defined in Fig. 2(b). The time-delay inputs, X_i $(i = 1, 2, \cdots, e_s)$, are undelayed or delayed D_t $(= \sum D_j + \Delta)$, where D is a delay unit and Δ is its increment $(j = 1, 2, \cdots, t-1; t = 1, 2, \cdots, r_s - 1)$. The time-delay speech inputs, X_i, will be multiplied by several weights, one for each delay and one for the undelayed input.

Note that two types of operations, namely, control flow and data flow, are used in this pipelined neural architecture. Master control unit not only gives each control sequence to the corresponding processing stage, but also arranges the time-delay speech input parameters of each frame as data flow input to the given computing window. Once the window is filled, the time-delay sequence input submatrix obtained is processed. Depending on the nature of the time-delay speech inputs, two different processing stages can be used in the architecture. These will be discussed in the next section.

III. Pipelined Architecture: Processing Stages

A. Parallel Processing Stage

In the pipelined neural system, a parallel processing stage can be defined in Fig. 3(a), where a window built by $m_s \times r_s$ elements is utilized to receive the input data flow from its previous stage, and m_{s+1} neurons are used to send the output results to the next stage in parallel. In other words, the input data flow is passed through the window and transformed by this stage to generate the corresponding output data flow. The widths of both data flows are defined as m_s and m_{s+1}, respectively. A new feature parameter obtained by each neuron in stage s can be represented as

$$y_i(s+1) = f\left(\sum W_i(s) X^T(s)\right) \tag{5}$$

where $s = 1, 2, \cdots, p+1$; $i = 1, 2, \cdots, m_{s+1}$; $W_i(s)$ is the ith weight matrix and $X(s)$ is the input submatrix given by the window in stage s.

B. Serial Processing Stage

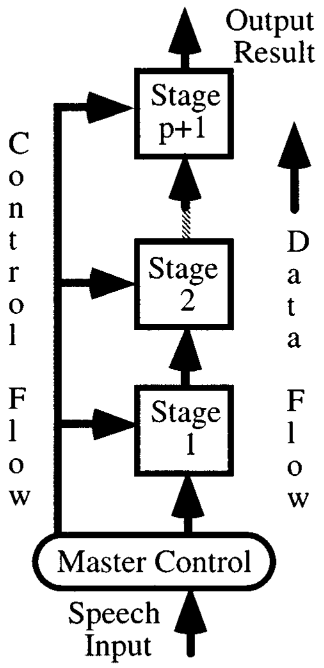
In this processing stage, a pipe with single parameter width is made by a chain of serial shifting elements [see Fig. 3(b)]. A window structure is designed to implement the transformation between stages. The $r_s - 1$ line delays, each $m_s - e_s$ shifting elements, are built to receive a serial stream of parameters and to form the required window $(e_s \times r_s)$, which are input to m_{s+1} neurons. Thus, the total $(e_s \times r_s) + (m_s - e_s)(r_s - 1) = m_s(r_s - 1) + e_s$ elements are needed in window structure. The neurons associated with the window are defined in (5) with their common output

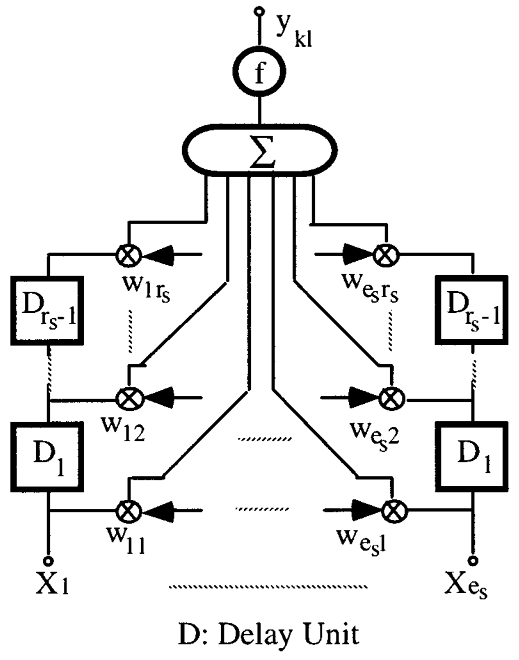
$$y(s+1) = OR(y_i(s+1)) \tag{6}$$

where $i = 1, 2, \cdots, m_{s+1}$; $s = 0, 1, \cdots, p+1$. Note that the output of each neuron can, in turn, be obtained as, either a single output, $y_i(s+1)$, or no output within a single clock interval. The output order in a cycle is: $y_1(s+1), \cdots, y_{m_{s+1}}(s+1), *_1, \cdots, *e_s - 1$, where $m_{s+1} + e_s - 1 = m_s$ and "$*$" indicates no output. There are a total of n_{s+1} $(= n_s - r_s + 1)$ processing cycles for each given speech input matrix, $m_s \times n_s$.

C. Building Elements

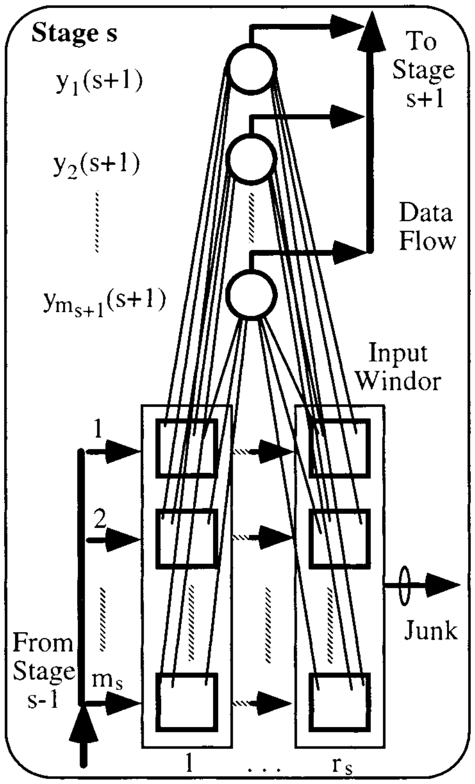
There are three kinds of building elements, including window, synapse and summing element, which are used in two different processing stages described before. A window element can be implemented by a regular shifting register and thus the following discussion will be focused on the other two building elements. Considering on-line backpropagation

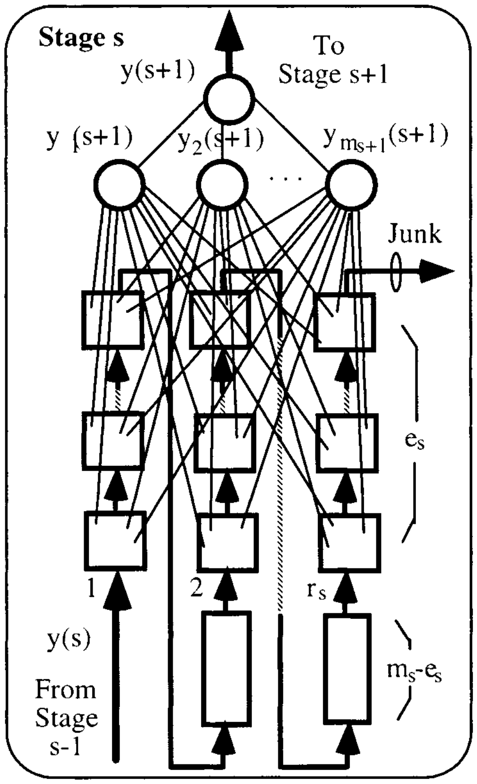




(a) (b)

Fig. 2. (a) Pipelined neural speech recognition system with $p+1$ processing stages and (b) time-delay neuron structure in the pipelined neural system.





(a) (b)

Fig. 3. Two kinds of processing stages in (a) parallel and (b) serial.

(BP) learning, which has successfully applied to the neural speech recognition systems, two processing phases, searching and learning, are defined in the building elements. They can be implemented by special feedforward and feedback paths, respectively.

1) Synapse Element: A synapse element is used to store and change weight values. It is mainly composed of a weight memory (W), two multipliers (I and II) and two selectors ($\otimes$ and $\oplus$), shown as in Fig. 4. A control clock, CLK, indicates the phase of the element. CLK $= 0$, means searching (or feed-

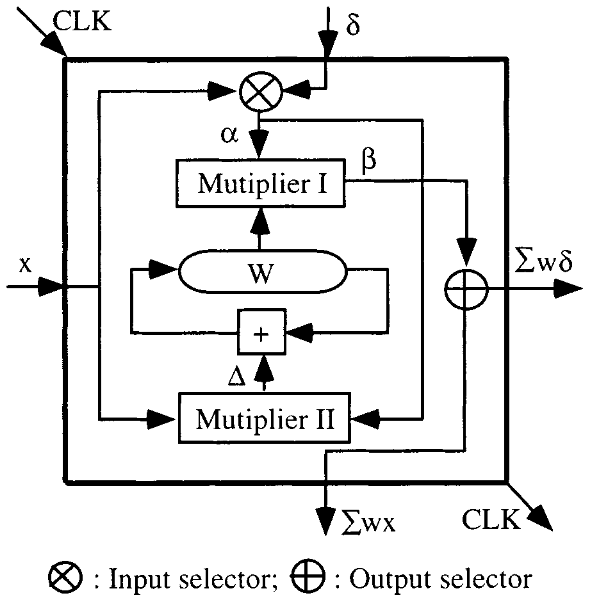


Fig. 4. Synapse building element structure.

forward) phase and CLK $= 1$, means learning (or feedback) phase. In this element, there are two data inputs (x and δ) and two outputs ($\sum wx$ and $\sum w\delta$), where $x/\sum wx$ work in CLK = 0, and $\delta/\sum w\delta$ in CLK = 1. Multiplier I can generate a common output

$$\beta = \alpha W \tag{7}$$

where α is the output of the input parameter selector, $\otimes$ which is represented as

$$\alpha = \begin{cases} x & \text{CLK = 0} \\ \delta & \text{CLK = 1.} \end{cases} \tag{8}$$

An output parameter selector, $\oplus$, can choose a correct output result of the element, i.e.,

$$\beta = \begin{cases} \sum wx & \text{CLK = 0} \\ \sum w\delta & \text{CLK = 1.} \end{cases} \tag{9}$$

Multiplier II is only designed to obtain the increment of the weight value when CLK $= 1$

$$\Delta = \eta\alpha x \tag{10}$$

where η is a gain. Using the arithmetic mechanism attached in the element, the increment, Δ, can be added to the weight, W, to generate a new weight value. In this way, when CLK $= 0$, the output of the element is $\sum wx$; otherwise, the output is $\sum w\delta$. Also, W is changed in terms of the following rule

$$W = W + \Delta = W + \eta\delta x. \tag{11}$$

2) Summing Element: This element is built to obtain two-direction accumulative results for both feedforward and feedback processing (see Fig. 5). It consists of two amplifiers and one multiplier. Two inputs (outputs), $\sum wx$ and $\sum w\delta'$ (δ and y), are from (to) the current stage and the next stage, respectively. Their input/output relations are $\sum wx/y$ and $\sum w\delta'/\delta$. When CLK = 0, the output of the element is represented as

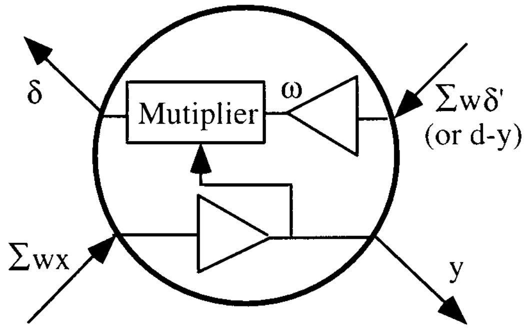


Fig. 5. Summing building element structure.

$$y = f\left(\sum wx\right) \tag{12}$$

and when CLK $= 1$, the output is

$$\delta = y(1-y)\omega = y(1-y)f\left(\sum w\delta'\right). \tag{13}$$

3) Connection Network: Three kinds of building elements can be easily implemented by the current VLSI technologies [13], [25], [26]. Using these simple building elements, a basic connection network in stage s can be designed as in Fig. 6, where the size of the time-delay computing window is defined as $m_s \times r_s$. There are a total of $m_s \times r_s \times m_{s+1}$ synapse elements and m_{s+1} summing elements used in the network. Obviously, the whole pipelined neural system can be implemented by cascading such regular connection networks.

IV. Systolic Array Implementation

A. Mapping Strategies

It is clear that the complexity of a computing window implementation stems not from the complexity of its nodes but rather from the multitude of ways in which a large collection of these nodes can interact. Therefore, an important task is to build highly parallel, regular and modular systolic arrays (SAs) that are attractive for VLSI techniques. Here we present different mapping strategies from pipelined architecture to SA with implementation efficiency as our goal.

1) Processing Mode Mapping: Here, we partition a pipelined neural system into some basic processing stages with time-delay window, each capable of performing an independent function. Often a processing stage represents a layer in the neural networks. The processing stages are implemented using a corresponding SA, which are then cascaded.

2) Computing Property Mapping: Each processing stage function is reduced to a recursive form which is implemented by the corresponding pipeline matrix in terms of some systolic rules. In practice, this mapping changes parallelism in place to parallelism in time.

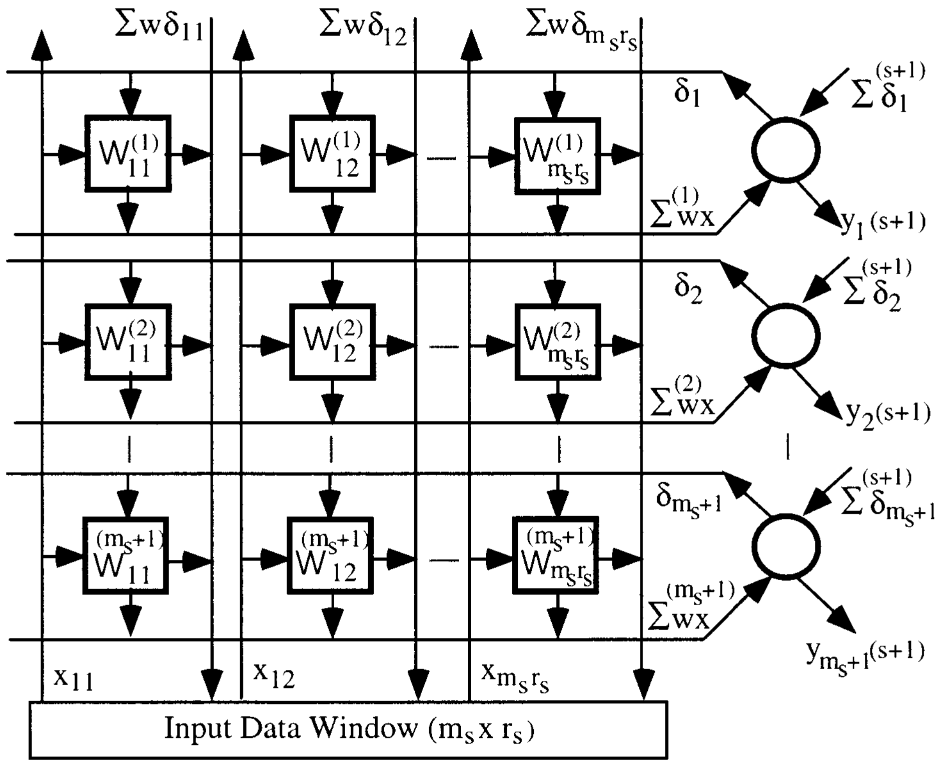


Fig. 6. Connection network with on-line learning in stage s.

3) Arithmetic Module Mapping: A basic operation in recursive arithmetic is implemented by a computing element. For example, a node is divided into two parts: forming a weighted sum of N inputs and passing the result through a nonlinearity. The weighted sum can easily be integrated by a two-dimensional (2-D) recursive matrix. To form the nonlinearity, a special element is defined which may be cascaded with the recursive matrix as a bound node of its output.

B. SA Structures: Computing Cell

Using the aforesaid mapping strategies two kinds of processing stages, in parallel and in serial, as obtained in Section III can be systematically implemented by the corresponding SAs (see Figs. 7 and 8). In both arrays, the line delays built by shifting elements are used to receive a data stream of parameters and to construct the window required. There are a total of $\{m_s(m_s-1)\}/2$ and $\{(2m_s-2e_s-r_s)(r_s-1)\}/2$ shifting elements in parallel processing SA and in serial processing SA, respectively. The adder arrays are built as the accumulators to compress the output results of the computing window. Obviously, some regular shifting registers and adders can implement the line delays and adder arrays.

Computing cells, defined in the both SAs, can be properly arranged to form each computing window in parallel or in serial. However, all of these computing cells have an identical structure with special feedforward and feedback paths. They are mainly composed of weight memory (W_i) $(i=1,2,\cdots,m_{s+1})$, adder $(\oplus)$ and multiplier $(\otimes)$. Three data inputs, x, $\{z_i\}$ and $\{\delta_i\}$, and their outputs, x', $\{Z_i\}$ and $\{\Omega_i\}$, are defined in the computing cell, where $\{z_i\}$ and $\{Z_i\}$ are used for CLK $=0$; $\{\delta_i\}$ and $\{\Omega_i\}$

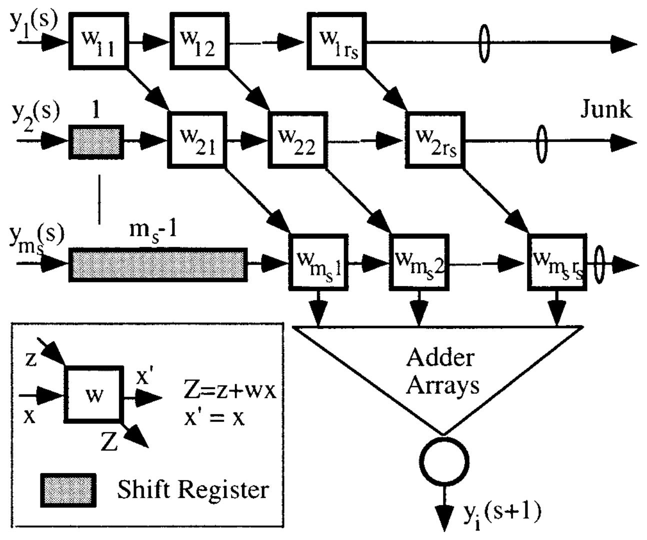


Fig. 7. Parallel data flow window computation.

for CLK $=1$. Note that each input (output) is transmitted by m_{s+1} data except x (x'). When CLK $=0$, the outputs of the cell in feedforward path are defined as: $Z=xw+z$ and $x'=x$; otherwise, the output as $\Omega=w\delta$. At the same time, W is changed in terms of the rule in (11).

It is evident that the SAs shown in Figs. 7, 8 are regular interconnected arrays using a set of computing cells, each performing some simple window computing, where the data flows in a rhythmic fashion with only local interconnects between cells. They can provide a good medium to implement the pipelined neural system in VLSI.

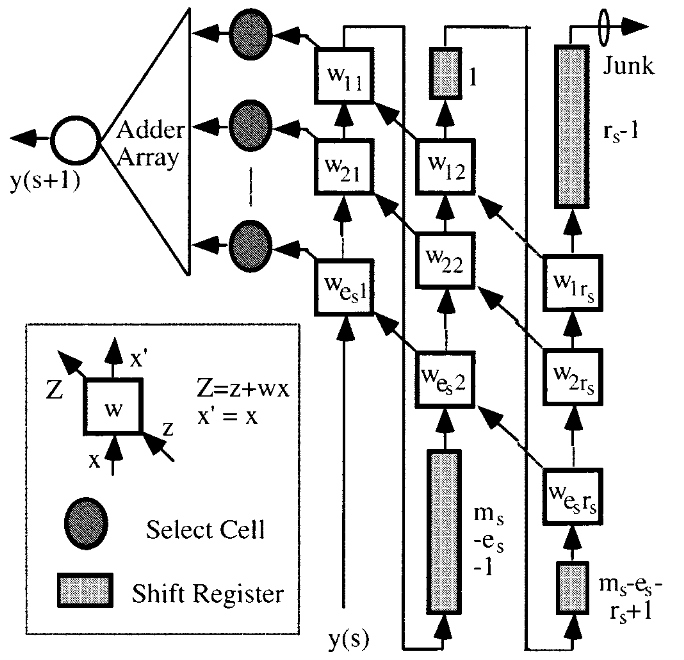


Fig. 8. Serial data flow window computation.

V. Speech Recognition Applications

In this section, we provide the results of our study on three types of neural systems using the parallel time-delay window computing in the pipelined neural architecture. The neural systems selected are motivated by speech recognition applications and they have been widely used [15]–[24].

A. Time-Delay Neural Network

Time-delay neural network (TDNN) is a neural system that can take into account the "dynamic nature of speech." It is used to represent temporal relationships between successive acoustic frames, while providing some invariance under time translation [19]. It has been demonstrated that the TDNN computing can provide excellent discrimination ability among speech sounds. Speech recognition performance obtained by using the TDNN has often exceeded that of many conventional approaches [20], [21].

The basic TDNN system is composed of an input layer, two hidden layers and an output layer [19]. Except the output layer, each layer has an $m_s \times n_s$ $(s = 1, 2, 3)$ matrix of memory elements, where $n_2 = n_1 - r_1 + 1$, $n_3 = n_2 - r_2 + 1$ and $m_3 = q$. The relation between the input layer and the 1st hidden layer (and also between first and second hidden layers; see Fig. 9) is represented as

$$y_{kl} = f\left(\sum \sum x_{ij} w_{i,j-l+1}^{(k)} + \theta^{(k)}\right) \tag{14}$$

where $l \le i \le m_s$ and $l \le j \le l + r_s + 1$. The TDNN computing can be implemented by the pipelined system with three parallel processing stages. Except the last stage without the data window, each input parameter matrix $(m_s \times n_s)$ in the first two stages is pipelined to pass through its window $(m_s \times r_s)$, $s = 1, 2$. When the window is filled by successive data flow, m_{s+1} new values of the parameters can be, in parallel, obtained and simultaneously fed into the window in the next stage. It is evident that there are m_{s+1} different weight matrices and n_{s+1} input data windows from stage s to stage $s+1$, and their sizes are equal to the size of the window in stage s, i.e., $m_s \times r_s$. Using this parallel window computing to implement the TDNN, only $m_s \times r_s$ window elements, instead of $m_s \times n_s$ (generally, $r_s \ll n_s$) elements, are needed in stage s.

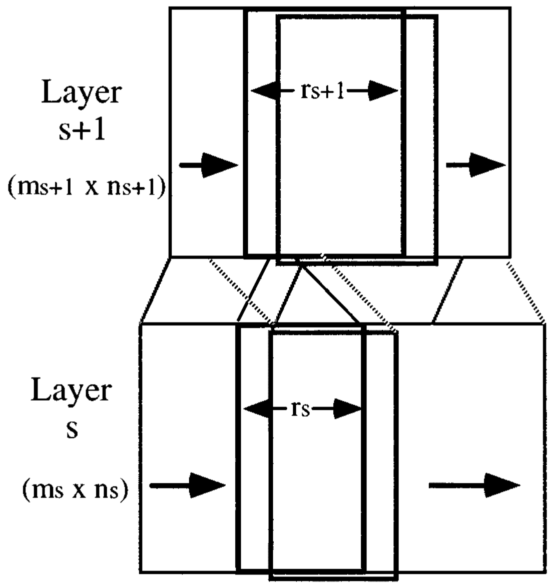


Fig. 9. Relation between layers for TDNN.

B. Block-Windowed Neural Network

Block-windowed neural network (BWNN) is based on windowing each layer of the neural network with overlaped local time-frequency windows. This neural system makes it possible to capture global features from the upper layers as well as precise local features from the lower layers. It is proved to be robust for speech sound variations in both frequency- and time-domains among different speakers [22].

The BWNN system is composed of an input layer, three hidden layers and an output layer [22]. Excepting the output layer, each layer has a $m_s \times n_s$ $(s = 1, 2, 3, 4)$ matrix of memory elements and their relation between layers satisfies

$$\begin{cases} m_{s+1} = m_s - e_s + 1 \\ n_{s+1} = n_s - r_s + 1 \end{cases} \tag{15}$$

where $m_4 = q$ and $e_s = r_s$, i.e., the length and the width of the submatrix in Layer s are the same (see Fig. 10). It is clear that the TDNN structure is a special example of the BWNN if $e_s = m_s$.

The use of the pipelined neural system to implement the BWNN involves four serial processing stages. Like the TDNN implementation, the last stage is the output stage without the data window. Each input matrix in the other stages can form an $m_s \times n_s$ $(s = 1, 2, 3)$ pipeline with the width of a single parameter and passes through its window $(e_s \times r_s)$, parameter by parameter. An output result obtained from stage s within a single clock interval is sent to the window in stage $s+1$ without any delay. This means that only an input window built by $e_s \times r_s$

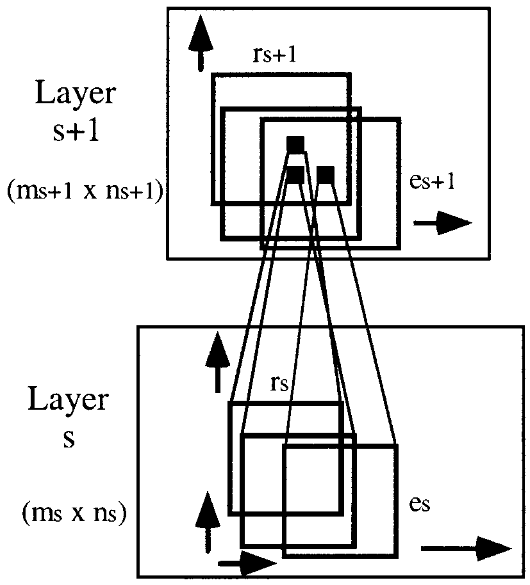


Fig. 10. Relation between layers for BWNN.

shifting elements and some line delays by $(m_s - e_s)(r_s - 1)$ shifting elements, i.e., the total $m_s(r_s - 1) + e_s$ window elements, instead of $m_s \times n_s$ elements (in general, $e_s \ll m_s$ and $r_s \ll n_s$), are needed in stage s.

C. Dynamic Programming Neural Network

Dynamic programming neural network (DNN) is proposed on the integration of multilayer neural network and dynamic programming based matching. Researchers have used DNN extensively in speaker-independent word recognition, and proved that it has excellent time normalization ability, flexible learning facility, expandability to continuous speech recognition, and high tolerance to the spectral pattern variation [23].

The DNN can be implemented by the pipelined neural system with three processing stages (see Fig. 11). An input pattern, $x_1, \cdots, x_i, \cdots, x_I$, is defined as a warping function $i = i(j)$ between input pattern time i and window element j, where $j = 1, 2, \cdots, J$. Without an input matrix with $I \times J$ memory elements [23], a window is built by $2 \times J$ window elements and J neurons are used in the first stage. When the input patterns, x_i and x_{i-1}, pass through the window, the corresponding output for each neuron can be represented as

$$y_{ij} = f(w_{j0}x_i + w_{j1}x_{i-1}) \tag{16}$$

where w_{j0} and w_{j1} are weighs from two window elements to neuron j.

In the second stage, a window with $J \times 1$ window elements is used to receive y_{ij} $(j = 1, 2, \cdots, J)$, in parallel. Each neuron in the stage is used as a multiplier, i.e.,

$$q_{ij} = f(w_j y_{ij}) = w_j y_{ij}. \tag{17}$$

The third stage is built by a serial processing structure. Its input data, q_{ij}, is arranged in a pipeline mode of a single parameter like $q_{iJ} \rightarrow \cdots \rightarrow q_{ij} \rightarrow \cdots \rightarrow q_{i1}$. In other words, the parallel outputs from the previous stage will be changed as the serial inputs to this stage. It is composed of a four-element window, two $J - 1$ line delays and a processing element (PE), shown in Fig. 12(a). The PE is designed by the standard dynamic programming algorithm [24]. Its initial condition is set at $g_{11} = q_{11}$, implemented by the external control. Then, the data is processed with

$$g_{ij} = q_{ij} + \max(g_{i,j-1}, g_{i-1,j-1}, g_{i-2,j-1}). \tag{18}$$

The PE shown in Fig. 12(b) implements this maximization problem. The PE consists of a tricomparator subnet for extracting the maximum of three analog inputs [31] and an adder. Given an input parameter, q_{ij}, an output of the PE, g_{ij}, can be obtained and fed into the window to generate the following new values. This process is continued until the total cumulating value, $Z = g(I, J)$, is reached. Such a process is represented in Fig. 13.

VI. Structure Analysis

For a given neural system, both the structure design and access time needed to solve the problem are two most important performance measures [13], [25]–[30]. In this section, we will analyze these measures for our pipelined neural architecture, where parallel processing stage defined in Fig. 3(a) and serial processing stage in Fig. 3(b) are referred to as type 1 and type 2, respectively. The way of selecting the property parameters for parallel time-delay window computing is also discussed in this section.

A. Structure Complexity

We choose a typical TDNN computing for comparison with our methodology. In Section V, it has been shown that the parallel time-delay window computing can implement TDNN and greatly reduce the memory elements in each layer of the neural networks to a small number of window elements in the processing stage. This is because only a limited window is connected to its next stage and the parameters shifted out from the window are discarded. Since the speech feature parameters are applied to each layer sequentially one frame at a time, this reduction of memory elements is feasible.

Note that both memory element used in traditional TDNN computing and window element in parallel window computing have the same hardware complexity because they are based on a regular register. In this way, we can perform the traditional TDNN computing by using the three kinds of building elements given in Section III. According to the basic TDNN definition [19], [20], it is assumed for the traditional computing that in Layer s $(s = 1, 2, \cdots, p{+}1)$, the number of window elements is taken as $m_s \times n_s$, the number of synapse elements as $m_s \times n_s \times m_{s+1}$, and the number of summing elements as $m_{s+1} \times n_{s+1}$.

In the parallel window computing, the numbers of window elements used in stage s for type 1 and type 2 (see Section III) have been given as $m_s \times r_s$ and m_s $(r_s - 1) + e_s$, respectively. Their measures for window elements can be defined as follows:

$$D_{type\ 1(s)} = \frac{m_s \times r_s}{m_s \times n_s} = \frac{r_s}{n_s} \tag{19}$$

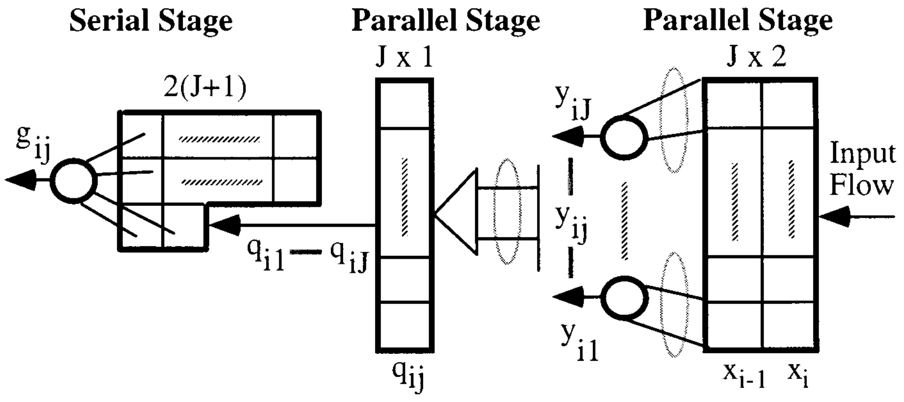


Fig. 11. Pipelined neural architecture for DNN implementation.

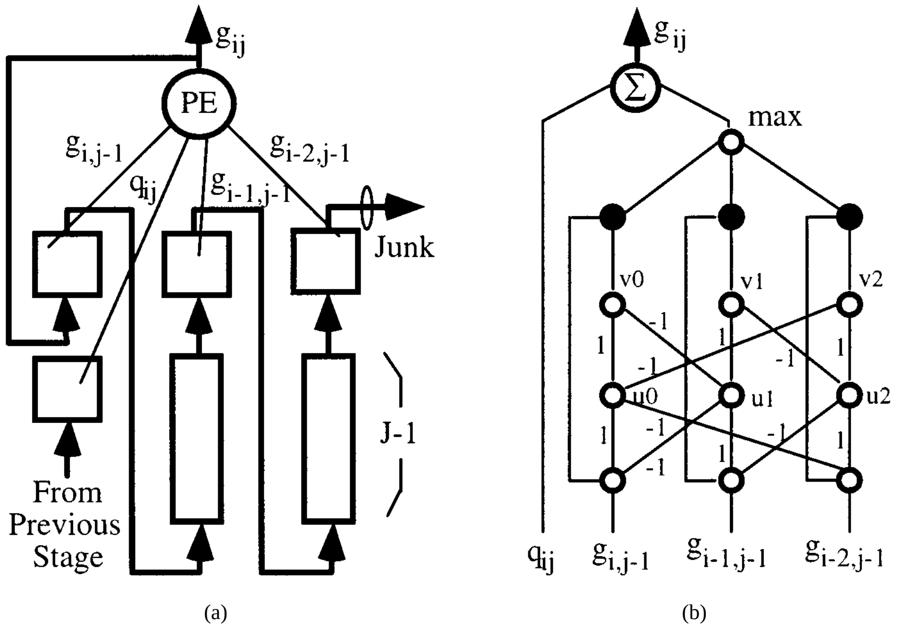


Fig. 12. (a) Serial processing stage for DNN and (b) its PE structure.

and

$$D_{type\ 2(s)} = \frac{m_s(r_s - 1) + e_s}{m_s \times n_s}. \tag{20}$$

Similarly, the numbers of both synapse and summing elements used in stage s for type 1 and type 2 can be obtained from Section III. Thus, the measures for synapse element are

$$Y_{type\ 1(s)} = \frac{m_s \times r_s \times m_{s+1}}{m_s \times n_s \times m_{s+1}} = \frac{r_s}{n_s} \tag{21}$$

and

$$Y_{type\ 2(s)} = \frac{e_s \times n_s \times m_{s+1}}{m_s \times n_s \times m_{s+1}} = \frac{e_s \times r_s}{m_s \times n_s}. \tag{22}$$

Both the numbers of summing elements used in stage s for type 1 and type 2 are m_{s+1}. Hence these two kinds of processing stages have the same measure, i.e.,

$$S_{type\ 1(s)} = S_{type\ 2(s)} = \frac{m_{s+1}}{m_{s+1} x n_{s+1}} = \frac{1}{n_{s+1}}. \tag{23}$$

It is evident that the measures of the entire system for three kinds of building elements are their mean of over each stage s.

As an example, the traditional TDNN computing for typical speech recognition applications has been described in [19], [20]: $m_1 \times n_1 = 16 \times 15$, $r_1 = 3$; $m_2 \times n_2 = 8 \times 13$, $r_2 = 5$; $m_3 \times n_3 = 3 \times 9$; $q = 3$. Then, in order to implement the basic TDNN, the measures of the first two type 1 processing stages in the neural pipelined system are: $D_{type\ 1} = Y_{type\ 1} = \frac{1}{2}\ (\frac{3}{15} + \frac{5}{13}) = 0.29$ and $S_{type\ 1} = \frac{1}{2}\ (\frac{1}{13} + \frac{1}{9}) = 0.09$. This means that three building elements in parallel time-delay window computing can be reduced by a factor of 3, 3, and 10, respectively.

The results of the above analysis are summarized in Table I. It indicates that the structure complexity for our parallel time-delay window computing is much less than that of the traditional TDNN computing.

B. Throughput Rate

The neural speech recognition systems are well suited to pipelining because of their multilayer networks as processors

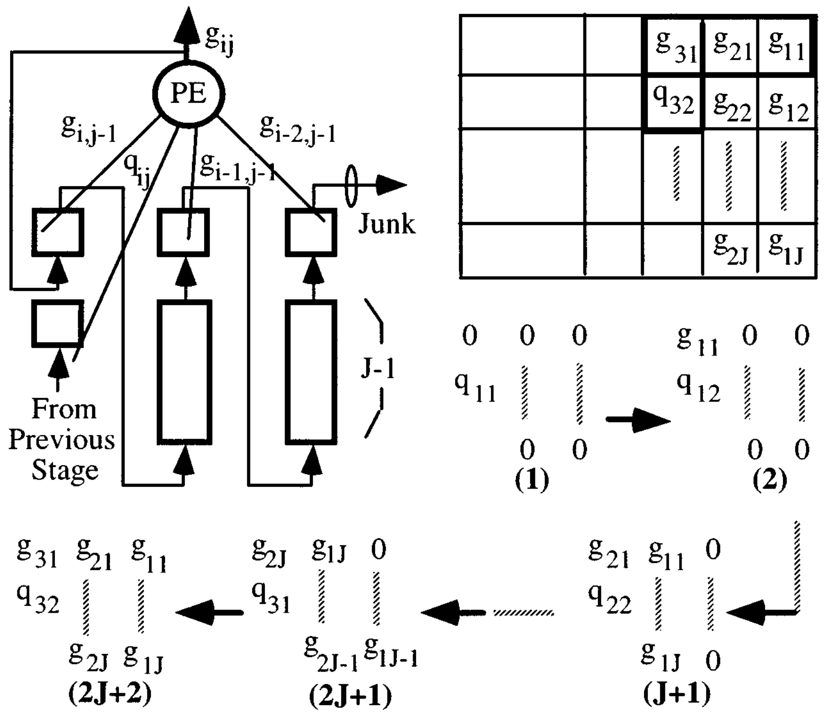


Fig. 13. Window process for DNN implementation.

TABLE I
COMPARISON BETWEEN TRADITIONAL TDNN COMPUTING AND PARALLEL WINDOW COMPUTING IN STAGE s

		Traditional TDNN Computing	*Parallel Window Computing*	
			Type 1	*Type 2*
Window Element	*Element #*	$m_s \times n_s$	$m_s \times r_s$	$m_s(r_s-1)+e_s$
	Complexity	1	$\frac{r_s}{n_s}$	$\frac{m_s(r_s-1)+e_s}{m_s \times n_s}$
Synapse Element	*Element #*	$m_s \times n_s \times m_{s+1}$	$m_s \times r_s \times m_{s+1}$	$e_s \times r_s \times m_{s+1}$
	Complexity	1	$\frac{r_s}{n_s}$	$\frac{e_s \times r_s}{m_s \times n_s}$
Summing Element	*Element #*	$m_{s+1} \times n_{s+1}$	m_{s+1}	m_{s+1}
	Complexity	1	$\frac{1}{n_{s+1}}$	$\frac{1}{n_{s+1}}$

of time-delay sequence patterns. In the pipelined system embedding parallelism or concurrency, the throughput rate can be fixed and it does not vary with the size of the problem grows, i.e.,

$$T = O(1). \tag{24}$$

Hence, a high throughput rate can be maintained in such pipelined neural systems, where the clock of the master control element is selected from the longest time delay among processing stages.

C. Window Parameter

Computing window in the pipelined neural system is not only an important component, but also an obvious feature which differs from other neural systems. The window size has a direct relation to the properties of the pipelined system, such as the number of window elements, Ω, and the computing time, Ψ. The smaller the window, the fewer is the number of window elements, and the longer is the computing time required.

In type 1, these two tradeoff properties for stage s are

$$\begin{cases} \Omega_{type\ 1} = r_s \times m_s, \\ \Psi_{type\ 1} = n_s - r_s + 1. \end{cases} \tag{25}$$

We define their product as

$$E(r_s) = \Omega_{type\ 1} \times \Psi_{type\ 1} = (r_s \times m_s)(n_s - r_s + 1). \tag{26}$$

To maximize the $E(r_s)$ function, take derivative with respect to window size r_s, i.e.,

$$E'(r_s) = m_s n_s - 2m_s r_s + m_s. \tag{27}$$

Let $E'(r_s) = 0$. The optimal size of window for type 1 [see Fig. 14(a)] can be then selected as

$$r_s = \frac{n_s + 1}{2}. \tag{28}$$

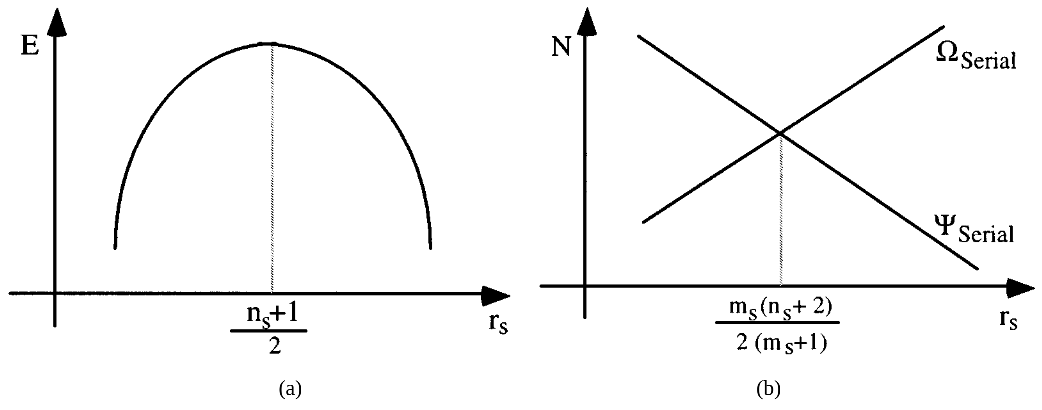


Fig. 14. (a) Window parameter selection for (a) type 1 and (b) type 2.

Note that this optimal window size is not a function of the length of the window, m_s. In the same way, the two tradeoff properties in type 2 can be written as:

$$\begin{cases} \Omega_{type\ 2} = (r_s - 1)m_s + r_s \\ \Psi_{type\ 2} = m_s n_s - (r_s - 1)m_s - r_s \end{cases} \tag{29}$$

where $r_s = e_s$, i.e., a square window is used. The size of the window can be selected directly from the relation $\Omega_{type\ 2} = \Psi_{type\ 2}$ [see Fig. 14(b)], which leads to

$$(r_s - 1)m_s + r_s = m_s n_s - (r_s - 1)m_s - r_s. \tag{30}$$

Hence, the choice of the window size for type 2 is

$$r_s = \frac{m_s(n_s + 2)}{2(m_s + 1)}. \tag{31}$$

VII. Conclusions

In this paper, a novel parallel structure for time-delay neural networks are used in speech recognition applications is presented. The effectiveness of the design has been illustrated by extracting a window computing model from the time-delay neural systems, developing the corresponding pipelined architecture with parallel or serial processing stages and comparing its performance with the traditional TDNN computing. Applying this parallel window to a typical time-delay neural network, it has been shown that the methodology can greatly reduce the structure complexity while maintaining a high throughput rate.

Acknowledgment

The authors would like to thank M. Elmasry and M. Kamel of the University of Waterloo for their valuable help.

References

[1] S. Furui, *Current Technology for Speech Recognition*. New York: IEEE Press, 1995.
[2] N. Morgan and H. Bourland, "Neural networks for statistical recognition of continuous speech," *Proc. IEEE*, vol. 83, pp. 742–770, May 1995.
[3] K. Chen, D. Xie, and H. Chi, "A modified HME architecture for text-dependent speaker identification," *IEEE Trans. Neural Networks*, vol. 7, no. 5, pp. 1309–1314, 1996.
[4] H. P. Campbell, "Speaker recognition: Tutorial," *Proc. IEEE*, vol. 85, pp. 1437–1463, Sept. 1997.
[5] K. Chen *et al.*, "Speaker identification using time-delay HMEs," *Int. J. Neural Syst.*, vol. 7, no. 1, pp. 29–43, 1996.
[6] G. Doddington, "Speaker recognition—identifying people by their voice," *Proc. IEEE*, vol. 73, no. 11, pp. 1651–1664, 1986.
[7] D. O'Shaugenessy, "Speaker recognition," *IEEE ASSP Mag.*, vol. 3, pp. 4–17, 1986.
[8] H. Bourland and C. J. Wellekens, "Links between Markov models and multilayer perceptrons," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 12, pp. 1167–1178, Dec. 1990.
[9] T. Matsui and S. Furui, "Speaker recognition technology," *NTT Rev.*, vol. 7, no. 2, pp. 42–48, 1995.
[10] D. P. Morgan and C. L. Scofield, *Neural Networks and Speech Processing*. Norwell, MA: Kluwer, 1991.
[11] L. M. Fu, *Neural Networks in Computer Intelligence*. New York: McGraw-Hill, 1994.
[12] C. Bishop, *Neural Networks for Pattern Recognition*. Oxford, U.K.: Oxford Univ. Press, 1995.
[13] M. I. Elmasry, *VLSI Artificial Neural Networks Engineering*. Norwell, MA: Kluwer, 1994.
[14] D. Zhang, *Parallel VLSI Neural System Designs*. Berlin, Germany: Springer-Verlag, 1998.
[15] R. P. Lippmann, "Review of neural networks for speech recognition," *Neural Comput.*, vol. 1, pp. 1–38, 1989.
[16] S. Furui and M. M. Sondhi, *Advances in Speech Signal Processing*. New York: Marcel Dekker, 1992.
[17] M. Koerner, *Speech Recognition*, M. Koener, Ed. Englewood Cliffs, NJ: Prentice-Hall, 1996.
[18] J. Junqua, *Robustness in Automatic Speech Recognition: Fundamentals and Applications*. Norwell, MA: Kluwer, 1995.
[19] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. J. Lang, "Phoneme recognition using time-delay neural networks," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 37, no. 3, pp. 328–339, 1989.
[20] A. Waibel, "Modolar construction of time-delay neural networks for speech recognition," *Neural Comput.*, vol. 1, pp. 38–46, 1989.
[21] K. J. Lang and A. Waibel, "A time-delay neural networks architecture for isolated word recognition," *Neural Networks*, vol. 3, pp. 23–43, 1990.
[22] H. Sawai, "Frequency-time-shift-invariant time-delay neural networks for robust continuous speech recognition," *Proc. Int. Conf. Acoust. Speech Signal Processing '91*, vol. S-2.1, pp. 45–48, 1991.
[23] H. Sakoe, R. Isotani, K. Yoshida, K. Iso, and T. Watanabe, "Speaker-independence word recognition using dynamic programming neural networks," *Speech Recognit.*, pp. 439–442, 1989.
[24] H. Sakoe and S. Chiba, "Dynamic programming algorithm optimization for spoken word recognition," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 26, no. 1, pp. 43–39, 1978.
[25] D. Zhang and M. I. Elmasry, "VLSI compressor design for digital neural network implementation," *IEEE Trans. Very Large Scale Integration Syst.*, vol. 5, no. 4, pp. 230–233, 1997.
[26] B. A. White and M. I. Elmasry, "The digi-neocognitron: A digital neocognitron neural network model for VLSI," *IEEE Trans. Neural Networks*, vol. 3, no. 1, pp. 73–85, 1992.

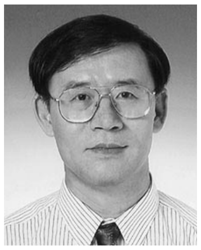
[27] J. B. Burr, "Digital neural network implementation," in *Neural Networks, Concepts, Applications, and Implementations*. Englewood Cliffs, NJ: Prentice-Hall, 1991, pp. 237–285.

[28] S. Y. Kung, *Digital Neural Networks*. Englewood Cliffs, NJ: Prentice-Hall, 1993.

[29] F. Y. Shih and J. Moh, "Implementing morphological operations using programmable neural networks," *Pattern Recognit.*, vol. 25, no. 1, pp. 89–99, 1992.

[30] Y. Takefuji, *Neural Network Parallel Computing*. Norwell, MA: Kluwer, 1994.

[31] F. Y. Shih and J. Moh, "Implementing morphological operations using programmable neural networks," *Pattern Recognit.*, vol. 25, no. 1, pp. 89–99, 1992.



David Zhang (M'92–SM'95) graduated in computer science from Peking University in 1974 and received the M.Sc. and Ph.D. degrees in computer science and engineering from Harbin Institute of Technology (HIT), China, in 1983 and 1985, respectively. He received his second Ph.D. in electrical and computer engineering at University of Waterloo, Ontario, Canada, in 1994.

From 1985 to 1988, he was Postdoctoral Fellow at Tsinghua University, China, and became Associate Professor at Academia Sinica, Beijing, China. In 1988, he joined the University of Windsor, Windsor, ON, Canada, as a Visiting Research Fellow in electrical engineering. Since 1995, he has been an Associate Professor with City University of Hong Kong and Hong Kong Polytechnic University. Currently, he is a Founder and Director of both Biometrics Technology Centres supported by URG/CRC, Hong Kong Goverment, and National Natural Scientific Foundation (NSFC) of China, respectively. He is also a Guest Professor and Ph.D. Supervisor. He is a Founder and Editor-in-Chief of the *International Journal of Image and Graphics* and an Associate Editor of *Pattern Recognition* and *International Journal of Pattern Recognition and Artificial Intelligence*. His research interests include automated biometrics-based identification, neural systems and applications, and parallel computing for image processing and pattern recognition. So far, he has published more than 150 papers including four books around his research areas and given over five keynotes or invited talks or tutorial lectures, as well as served as program/organizing committee members and session co-chairs at international conferences in recent years.He has developed some applied systems

He is listed in the *1999 Marquis Who's Who in the World* (16th ed.) and has received several project awards.

Sankar K. Pal (M'81–SM'84–F'93) received the M.Tech. and Ph.D. degrees in radio physics and electronics from the University of Calcutta, Calcutta, India, in 1974 and 1979, respectively. In 1982, he received another Ph.D. degree in electrical engineering along with the DIC from Imperial College, University of London, London, U.K.

He is a Distinguished Scientist and Founding Head of the Machine Intelligence Unit, Indian Statistical Institute, Calcutta. He was with the University of California, Berkeley, and the University of Maryland, College Park, during 1986–1987 as a Fulbright Postdoctoral Visiting Fellow. He was with the NASA Johnson Space Center, Houston, TX, during 1990–1992 and a Guest Investigator under the NRC-NASA Senior Research Associateship program in 1994. He was with the Hong Kong Polytechnic University in 1999 as a Visiting Professor. His research interests includes pattern recognition, image processing, soft computing, neural nets, genetic algorithms, and fuzzy systems. He is a co-author of six books including *Fuzzy Mathematical Approach to Pattern Recognition* (New York: Wiley, 1986) and *Neuro-Fuzzy Pattern Recognition: Methods in Soft Computing* (New York: Wiley, 1999). He is an Associate Editor for *Pattern Recognition Letters*, *Neurocomputing*, *Applied Intelligence*, *Information Sciences*, *Fuzzy Sets and Systems*, *Fundamenta Informaticae*, and the *International Journal of Approximate Reasoning*.

Dr. Pal is a Fellow of the Third World Academy of Sciences, Italy, and all the four National Academies for Science/Engineering in India. He served as a Distinguished Visitor of the IEEE Computer Society for the Asia-Pacific Region during 1997–1999. He received the 1990 S. S. Bhatnagar Prize, the 1993 Jawaharial Nehru Fellowship, the 1993 Vikram Sarabhai Research Award, the 1993 NASA Tech Brief Award, the 1994 IEEE TRANSACTIONS ON NEURAL NETWORKS Outstanding Paper Award, the 1995 NASA Patent Application Award, the 1997 IETE-Ram Lal Wadhwa Gold Medal, the 1998 Om Bhasin Foundation Award, and the 1999 G. D. Birla Award for Scientific Research. He was an Associate Editor for the IEEE TRANSACTIONS ON NEURAL NETWORKS (1994–1998), and is a Member of the Executive Advisory Editorial Board for the IEEE TRANSACTIONS ON FUZZY SYSTEMS and a Guest Editor of many journals including IEEE COMPUTER.