Open access • Journal Article • DOI:10.1109/TASSP.1986.1164918

Systolic architectures for connected speech recognition — Source link

François Charot, P. Frison, Patrice Quinton

Published on: 01 Aug 1986 - IEEE Transactions on Acoustics, Speech, and Signal Processing (Springer, Berlin, Heidelberg)

Topics: Dynamic time warping, Systolic array, Feature (machine learning), Blossom algorithm and Probabilistic logic

Systolic architectures for connected speech recognition

François Charot, Patrice Frison, Patrice Quinton

IRIA

CENTRE DE RENNES

IRISA

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France
Tél.(3)954 90 20

Rapports de Recherche

N° 332

SYSTOLIC ARCHITECTURES FOR CONNECTED SPEECH RECOGNITION

François CHAROT
Patrice FRISON
Patrice QUINTON

Septembre 1984

IRISA

INSTITUT DE RECHERCHE EN INFORMATIQUE
ET SYSTÈMES ALÉATOIRES

Campus Universitaire de Beaulieu
Avenue du Général Leclerc
35042 - RENNES CÉDEX
FRANCE
Tél. : (99) 36.20.00
Télex : UNIRISA 95 0473 F

ARCHITECTURES SYSTOLIQUES POUR LA RECONNAISSANCE DE MOTS CONNECTES (en anglais)

François CHAROT
Patrice FRISON
Patrice QUINTON

Publication Interne n° 234
40 pages
Août 1984

RESUME : On décrit des architectures systoliques pour deux méthodes de reconnaissance de mots connectés. La première méthode est fondée sur l'algorithme de comparaison dynamique qui est appliqué directement sur les données acoustiques. La seconde méthode est un algorithme de comparaison probabiliste qui nécessite que la phrase d'entrée ait été prétraitée par un analyseur phonétique. On montre que ces deux méthodes peuvent être mises en oeuvre sur des architectures systoliques à une ou deux dimensions. Les avantages de chaque mise en oeuvre sont discutés. On décrit en outre l'architecture d'un circuit programmable de 12 000 transistors, réalisé en technologie NMOS, qui peut être utilisé comme processeur élémentaire pour chacune de ces mises en oeuvre.

SUMMARY : Systolic arrays for two connected speech recognition methods are presented. The first method is based on the dynamic time warping algorithm which is applied directly on acoustic feature patterns. The second method is the probabilistic matching algorithm which requires that the input sentence be preprocessed by a phonetic analyser. It is shown that both methods may be implemented on either a two-dimensional or a linear systolic array. Advantages of each of these implementations are discussed. The architecture of a 12 000 transistors programmable NMOS prototype IC which can be used as the basic processor of these systolic arrays is presented.

SYSTOLIC ARCHITECTURES FOR CONNECTED SPEECH RECOGNITION (*)

Francois CHAROT
Patrice FRISON
Patrice QUINTON

IRISA, Campus de Beaulieu,
35042 RENNES-CEDEX
FRANCE

July 1984

ABSTRACT

Systolic arrays for two connected speech recognition methods are presented. The first method is based on the dynamic time warping algorithm which is applied directly on acoustic feature patterns. The second method is the probabilistic matching algorithm which requires that the input sentence be preprocessed by a phonetic analyzer. It is shown that both methods may be implemented on either a two-dimensional or a linear systolic array. Advantages of each of these implementations are discussed. The architecture of a 12000 transistors programmable NMOS prototype IC which can be used as the basic processor of these systolic arrays is presented.

(*) This work has been partly funded by a grant of the CNET in Lannion (French Telecommunication Research Center).

SYSTOLIC ARCHITECTURES FOR CONNECTED SPEECH RECOGNITION

Francois CHAROT
Patrice FRISON
Patrice QUINTON

IRISA, Campus de Beaulieu,
35042 RENNES-CEDEX
FRANCE

1. INTRODUCTION

Speech recognition is a computationally intensive task which can be handled on conventional architectures only for small vocabularies. The use of pipeline and parallel architectures organizations increases the speed of the algorithm of one order of magnitude, thus making realistic recognition of much larger vocabularies and broadening the range of application of vocal input.

Over the past few years, numerous attempts to map speech recognition algorithms on parallel architectures or to design dedicated parallel architectures supporting speech tasks have been reported [1]. Advances in VLSI have shown to be a determinant factor in this evolution as VLSI makes it feasible to implement complex parallel architectures on a few integrated circuits. Among the various parallel organizations that have been considered for that purpose, systolic array is a particularly appealing structure. A systolic array [2] is a special-purpose architecture made out of simple processing elements organized as a regular network. Processors are locally connected, operate synchronously, and data circulate throughout the network in a pipeline fashion. The regular structure and operation of systolic arrays are particularly well suited for VLSI implementation hardware, since the design time can be dramatically reduced by using several instances of the same cell, and the local communications help solving the problem of long wires.

Systolic arrays for speech recognition have been described recently. Weste, Burr and Ackland [3] present an orthogonal array of processors (40×40) that can support the dynamic time warping algorithm. The performances of this array permit real time isolated word recognition of a 20,000 word vocabulary. They also describe the VLSI implementation of the basic processor of this array. Yoder and Siegel [4] present various systolic schemes for dynamic time warping, including the use of a linear array. More recently, Feldman et al. [5] consider a wafer scale implementation of a two-dimensional systolic array for connected speech recognition.

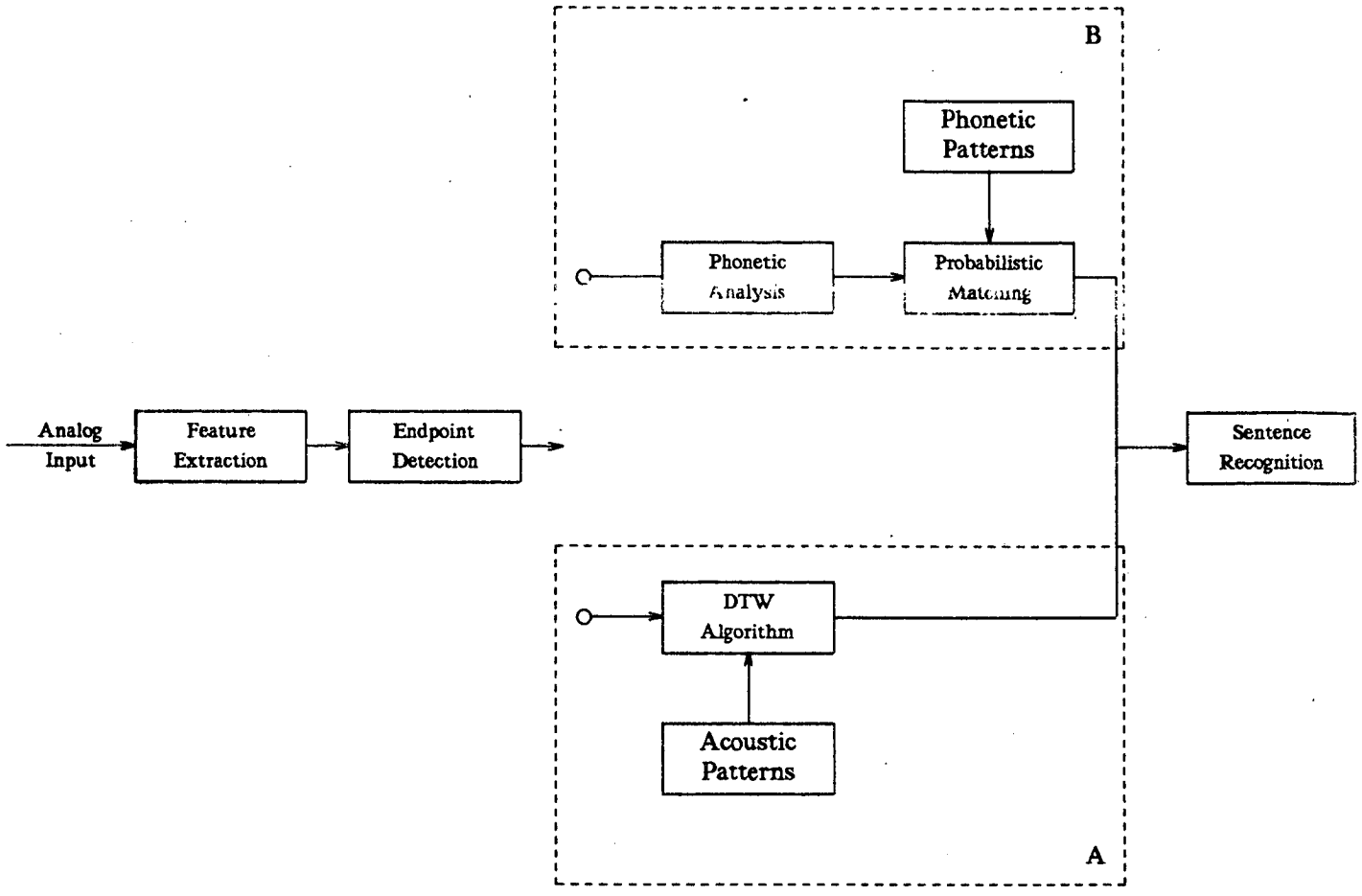

Fig. 1: Block diagram of a Connected Speech Recognition system:
A) using Dynamic Time Warping
B) using Probabilistic Matching

This paper is based on an effort undertaken since the end of 1980 at IRISA to explore the potential of systolic architectures for connected speech recognition [6]-[9]. The algorithm that was originally considered in [6] is a probabilistic matching algorithm based on Bahl and Jelinek linguistic decoder [10]. The main difference with the dynamic time warping approach is that the speech signal must be preprocessed before applying the recognition, in order to recognize the phonemes in the speech. It turns out that both dynamic time warping and probabilistic matching algorithms have the same structure and may thus be implemented on the same kind of systolic array. Our purpose in this paper is to present and discuss various systolic implementations for both methods. Section 2 presents the general organization of a connected speech recognition system and describes the dynamic time warping algorithm and the probabilistic matching. Section 3 shows how these algorithms may be implemented on a two-dimensional systolic array. Section 4 considers algorithms for a linear systolic array. In section 5, we describe the API89 chip, which is a prototype programmable chip that can be used as a basic processor for the various systolic organizations presented. Finally, section 6 compares the various solutions proposed.

2. SPEECH RECOGNITION ALGORITHMS

Fig. 1 presents together the block-diagram of two connected speech recognition methods. The first one (Fig. 1A) is based on the DTW algorithm, and the second one (Fig. 1B) on a probabilistic matching algorithm. In both methods, the analog signal which results of an utterance is filtered, and acoustic features are extracted at constants intervals. Depending on the acoustic analysis method which is used, these features may be band energies, formant frequencies, cepstral coefficients, or linear prediction coefficients. Each vector of features, called a **frame**, encodes usually 20ms of speech signal and contains 8 to 15 values. Endpoints of the utterance are then determined by examining variations in the energy of the speech signal. At this point, the pronounced utterance is represented by a sequence of frames called the **acoustic test pattern.** Two different approaches are then possible. In the first one (Fig. 1A), reference words are memorized as acoustic patterns. Distance measures between reference words and every subsequence of the test pattern are calculated using the dynamic time warping algorithm. These distances are then combined in order to retrieve the original sentence. The other approach (Fig. 1B) consists in preprocessing the acoustic test pattern in order to encode it as a string of phonemes called the **phonetic test pattern.** The reference words, also represented as strings of phonemes, are then matched against the phonetic test pattern resulting in probabilities that are then used for the sentence recognition.

From the point of view of recognition accuracy, the DTW approach is superior to the probabilistic matching because the phonetic analysis that must be carried out before the probabilistic matching is a very inaccurate process. This explains why up to now only the DTW algorithm has been considered for practical application of speech recognition. However, in the long term, the second approach presents the advantage of being less computation intensive and of needing less memory to store the reference words.

In the following, we describe in more detail the algorithms underlying each of these approaches. The DTW algorithm is first presented. Then we describe the connected word

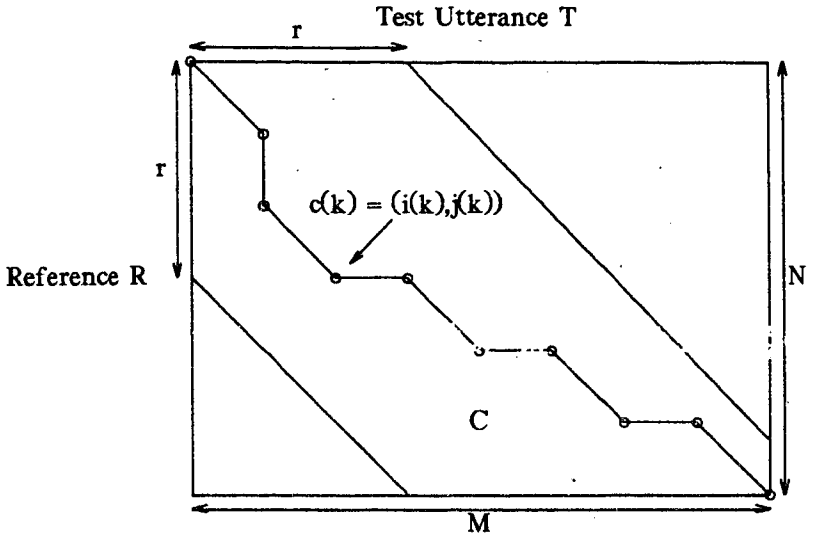

Fig. 2: Illustration of the Dynamic Time Warping algorithm.

recognition algorithm based on DTW. Finally, the probabilistic matching algorithm is explained.

2.1. THE DYNAMIC TIME WARPING ALGORITHM

The dynamic time warping algorithm is an application of the dynamic programming principle [10] to speech recognition. The following description is based essentially on the paper of Sakoe and Chiba [11].

Let $R = R(1) \cdots R(N)$ be a **reference word** and $T = T(1) \cdots T(M)$ be a **test utterance.** Given a pattern R, (word or connected word sequence), we shall denote as $R(i:j) = R(i) \cdots R(j)$ the subpattern of R from frame i to frame j and by $|R|$ the number of frames of R. Let $d(i,j) = \| R(i)-T(j) \|$ denote a distance between frame i of R and frame j of T . This distance may be the Chebischev distance, the Euclidean distance, or the log spectral distance depending on the type of acoustic feature considered.

Let C be a parametric curve of the plane defined by $C(k) = (i(k),j(k))$, $k=1 \cdots K$, where $C(1) = (1,1)$ and $C(K) = (N,M)$ (Fig.2). The dynamic time warping distance between R and T is defined by:

$$D(R,T) = \min_{C} \left[\frac{\sum_{k=1}^{K} d(c(k)) \, . \, w(k)}{\sum_{k=1}^{K} w(k)} \right] \tag{1}$$

where $w(k)$ are weighting coefficients.

Various recurrent schemes have been proposed to compute $D(R,T)$ depending on restrictions made on the curve C. In the following, we shall only consider one of the most commonly used recurrent scheme [11]. The curve C is constrained to have a slope p such that $\frac{1}{2} \leqslant p \leqslant 2$ and the weighting coefficients $w(k)$ are given by:

$$w(k) = j(k) - j(k-1) \tag{2}$$

We have then:

$$\sum_{k=1}^{K} w(k) = M \tag{3}$$

Such a DTW-distance is then said to be **assymetric.** Equation (1) may be solved by solving the following recurrence:

$$D(i,j) = \min \begin{cases} D(i-1,j-2) + d(i,j-1) + d(i,j) \\ D(i-1,j-1) + d(i,j) \\ D(i-2,j-1) + \dfrac{d(i-1,j) + d(i,j)}{2} \end{cases} \tag{4}$$

with the following initial conditions:

$$D(0,0) = 0 \; ; \; D(i,j) = \infty \quad \text{if } i \leqslant 0 \text{ and } j \geqslant 1 \text{ or if } i \geqslant 1 \text{ and } j \leqslant 0 \tag{5}$$

It results immediately from (1) and (3) that:

$$D(R,T) = \frac{D(N,M)}{M} \tag{6}$$

Note that normalization by the coefficient M is not necessary if the comparisons are made only between distances involving the same test pattern. Since this will be the case in the remaining of this paper, we shall confuse $D(R,T)$ and $D(N,M)$. Finally, a very common heuristics is that values $D(i,j)$ are computed only for the points (i,j) which lie in the band defined by:

$$| i - j | \leqslant r \tag{7}$$

where r is a given constant. This restriction avoids unreasonable stretching or compression of the reference pattern, and reduces the amount of computations to be carried out.

2.2. CONNECTED SPEECH RECOGNITION USING DTW

The DTW algorithm may be used either for isolated word recognition, or as a basis for connected word recognition. We shall only consider the later case here. Algorithms for connected word recognition have been described by Banatre et al. [7], Bridle et al. [13], Myers and Rabiner [14], and Sakoe [15]. The algorithms described in [14] and [15] find the best matching connected word sequence considering successively sequences of one word, then two words, etc. The results are then compared and the best connected word sequence is determined. In [7] and [13], the best connected word sequence is found in a single pass of the algorithm. In [13], the DTW algorithm and the connected word algorithm are merged, which results in a very efficient but memory consuming algorithm. In [7], the DTW algorithm and the connected word algorithm are done separately. The following description is based on [7].

Let T be the pattern resulting from the acoustic analysis of a sequence of words taken from a given vocabulary. We denote as V the number of reference words of the vocabulary, and denote as R_ν the ν^{th} reference word. The length of R_ν is denoted as N_ν. (The index ν will be omitted when understood by the context). We call **super-reference** and denote as R^S the pattern obtained by the concatenation of a finite number of words R_ν of the vocabulary. $\|R^S\|$ denotes the number of words of R^S (not to be confused with $|R^S|$, which is the number of frames of R^S).

With the above notations, the connected speech recognition algorithm consists, given a test sentence T, in finding the super-reference R^S which minimizes the distance $D(R^S,T)$.

This process consists in two steps. First, one computes the quantity:

$$D^* = \min_{R^S} D(R^S,T) \tag{8}$$

Then one finds out the sequence of words whose concatenation achieves the distance D^*.

Let $D^+(b,e)$ be the minimal distance between any reference word and a subpattern $T(b:e)$ of T, i.e:

$$D^+(b,e) = \min_{1\leq v\leq V} D(R_v,T(b:e)) \tag{9}$$

We denote as $v^+(b,e)$ the number of the reference pattern which minimizes (9). Let $D^*(e)$ be the minimum distance between any super-reference and $T(1:e)$, defined by:

$$D^*(e) = \min_{R^S} D(R^S,T(1:e)) \tag{10}$$

Finally, let $D^*_L(e)$ be the distance between a L-word length super-reference and $T(1:e)$. Since the recurrence scheme of equation (4) is considered, a L-word super-reference can match only a pattern having at least L frames. As a consequence, we have:

$$D^*(e) = \min_{1\leq L\leq e} D^*_L(e) \tag{11}$$

On the other hand, since the DTW distance is assymetric, $D^*_L(e)$ obey the following recurrence equation:

$$D^*_L(e) = \min_{1\leq b\leq e} \left[D^*_{L-1}(b-1) + D^+(b,e) \right] \tag{12}$$

Substituting (12) into (11) gives:

$$D^*(e) = \min_{1\leq L\leq e} \left[\min_{1\leq b\leq e} \left[D^*_{L-1}(b-1) + D^+(b,e) \right] \right] \tag{13}$$

By inverting the two minimum operations, we obtain:

$$D^*(e) = \min_{1\leq b\leq e} \left[\min_{1\leq L\leq e} \left[D^*_{L-1}(b-1) + D^+(b,e) \right] \right]$$

or, equivalently:

$$D^*(e) = \min_{1\leq b\leq e} \left[\left[\min_{1\leq L\leq e} D^*_{L-1}(b-1) \right] + D^+(b,e) \right]$$

which gives finally:

$$D^*(e) = \min_{1\leq b\leq e} \left[D^*(b-1) + D^+(b,e) \right] \tag{14}$$

In this equation, the quantity $D^+(b,e)$ may be obtained by applying the DTW algorithm given by equation (4). For a given value b and a given word R_v, a single application of the

algorithm allows all the quantities $D(R_v,T(b:e))$ to be computed for all values e lying in the interval $[e_1,e_2]$ where

$$e_1 = b - 1 + N_v - r$$

$$e_2 = b - 1 + N_v + r \tag{15}$$

According to equation (9) a first minimization over the index v gives then the distance $D^+(b,e)$. Finally, a second minimization over b by applying equation (14) gives the final result.

In order to retrieve the string of reference word numbers that minimizes $D^*(e)$, it is sufficient to keep track during the calculation of (14) of the index $b^+(e)$ which achieve the minimum as well as the number $v^+(b,e)$ of the word which achieves the minimum distance $D^+(b,e)$. The string of word numbers $v^*(e)$ which minimizes $D^*(e)$ is then obtained by the following recurrence:

$$v^*(e) = v^*(\, b^+(e)\,) \,.\, v^+(\, b^+(e) + 1\,,\, e) \tag{16}$$

where $x.y$ denotes the concatenation of strings x and y.

2.3. PROBABILISTIC SPEECH RECOGNITION

As mentioned earlier, another way to recognize speech is to preprocess the input data in order to identify the phonemes that have been pronounced. This process, called **phonetic analysis** is independent of the vocabulary chosen for the application, and reduces by approximately a factor of 5 the amount of information to be processed later on. Fig. 3 gives an example of the result of the phonetic analysis. The ideal transcription of the French sentence "liste des connecteurs" is presented together with the results of the phonetic analysis, as performed by the KEAL speech understanding system [16]. Each frame $T(i)$ contains a few phoneme candidates (three to five usually), with each of which is attached a probability (the probabilities are not represented on the figure for the sake of clarity). We denote as $x(i,k)$ the k^{th} candidate phoneme of frame $T(i)$ and $p(i,k)$ the probability associated with it.

Let R_v be the words of the vocabulary, where the frames of R_v are now the phonemes of the ideal phonetic transcription of the word. The connected speech recognition algorithm consists in retrieving the super-reference R^s that is most likely to have produced the test pattern T. Such a process may be carried out by the DTW algorithm provided a distance is defined between frames of the reference words and of the test utterance. However, the information lost during the phonetic analysis makes it impossible to achieve a high recognition accuracy in this way.

Another way to proceed is to describe the behavior of the phonemic analyzer more accurately using the concept of Probabilistic Finite State Machine (PFSM) introduced by Bahl and Jelinek [10]. To each phoneme y is associated a PFSM (Fig. 4) which describes how the phonemic analyzer is likely to deal with it. The PFSM has two states $S(0)$ and $S(1)$. Before reading y, it is is state $S(0)$, and may choose between three possibilities, namely **insertion, confusion** or **omission.** With each of these three possibilities is associated a probability

denoted respectively $P_i(y)$, $P_c(y)$, and $P_o(y)$. If the insertion is chosen, the PFSM remains in state $S(0)$, and produces a phonetic label x, with a conditional probability $q_i(x|y)$. In the case of a confusion, the PFSM reads the phoneme y and outputs x with conditional probability $q_c(x|y)$. Finally, if the omission is chosen, the PFSM skips the phoneme y with probability $P_o(y)$. In order for the model to be consistent, we must have the following relationships for every phoneme y:

$$P_i(y) + P_c(y) + P_o(y) = 1$$

$$\sum_x q_i(x|y) = 1$$

$$\sum_x q_o(x|y) = 1$$

These probability distributions can be estimated from the actual results of the phonemic analyzer.

From the elementary PFSM's associated with each individual phonemes, it is possible to modelize the behavior of the phonetic analysis when dealing with a reference word $R_v = R_v(1) \cdots R_v(N_v)$ by concatenating the PFSM's associated to $R_v(1)$, ... , $R_v(N_v)$ (Fig. 5). It is convenient to assume that every reference word ends with a special marker denoted] whose PFSM is such that $P_i(]) = P_c(]) = 0$, which prevents the PFSM to produce any result when reading this special marker.

The recognition process may be carried out from this model as follows. Consider the likelihood that a test pattern T has been produced by analyzing a given reference $R = R(1). \cdots .R(N)$. Let us denote by $L(i,j)$ the probability for the PFSM to enter state $S(i)$ after having produced $T(1{:}j)$. The PFSM may enter $S(i)$ after either an insertion of any candidate phoneme of $T(i)$ before reading $R(j)$, or after confusing $R(j)$ with one of the candidate phoneme of $T(i)$, or finally after skipping $R(j)$. As a consequence, $L(i,j)$ is given by the following recurrence equations:

$$L(i,j) = L_i(i,j) + L_c(i,j) + L_o(i,j) \tag{17}$$

$$L_i(i,j) = L(i,j-1) \times P_i(\, R(i+1)\,) \times \sum_k p(j,k)\, q_i(\, x(j,k) \mid R(i+1)) \tag{18}$$

$$L_c(i,j) = L(i-1,j-1) \times P_c(\, R(i)\,) \times \sum_k p(j,k)\, q_c(\, x(j,k) \mid R(i)) \tag{19}$$

$$L_o(i,j) = L(i-1,j) \times P_o(\, R(i)\,) \tag{20}$$

where $L_i(i,j)$, $L_c(i,j)$ and $L_o(i,j)$ denote respectively the probability for the PFSM to enter state $S(i)$ after inserting $T(j)$, confusing $T(j)$ and $R(i)$, or missing $R(i)$. Equation (17) is valid when $1 \leqslant i \leqslant N$ and $1 \leqslant j \leqslant M$. If we add the following conditions:

Fig. 3: Example of phonetic transcription for the french sentence "liste des connecteurs".

Pronounced sentence	liste des connecteurs														
Ideal phonetic translation	X_1	l X_2	i X_3	s X_4	t	œ X_5	d X_6	e X_7	k X_8	ø X_9	n X_{10}	$\tilde{\varepsilon}$ X_{11}	k	t X_{12}	œ X_{13}	r
Output of the phonetic analyzer	p t k	l j	i e	s f ʃ		œ ø	b d g	e y ɛ	p t k	ø œ $\tilde{\varepsilon}$	n v μ	ɛ $\tilde{\varepsilon}$ e		p t k	ø œ	

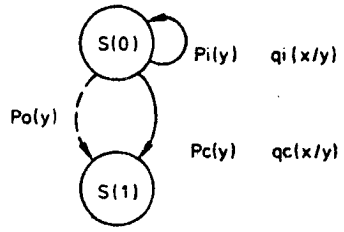

Fig. 4 PFSM associated with the phoneme y

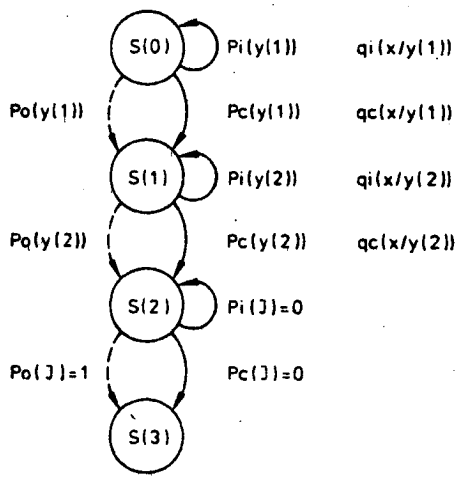

Fig. 5 PFSM associated with the word $Y = y(1)\ y(2)$

$$L_c(i,0) = 0 \quad \text{if} \quad i>0$$

$$L_c(0,j) = 0 \quad \text{if} \quad j>0$$

$$L_c(0,0) = 1 \tag{21}$$

$$L_j(i,0) = 0 \quad \forall i$$

$$L_o(0,j) = 0 \quad \forall j$$

then (17) is still valid if $i = 0$ or $j = 0$.

As in the case of the DTW algorithm, one can restrict the computation of $L(i,j)$ to the points (i,j) which lie into a band of width r, i.e. such that $| i - j | \leqslant r$.

The above probabilistic model may be adapted to connected speech recognition, in a very similar way as explained in section 2.2 for the DTW algorithm. Suppose now that T has been produced by the phonetic analysis of a super-reference R^S. Denote as $L^*(e)$ the maximum probability of a super-reference R^S given a test $T(1{:}e)$, and as $L^+(b,e)$ the maximum probability of a single reference word given the test $T(b,e)$. By a similar reasoning, $L^*(e)$ can be estimated by:

$$L^*(e) = \max_{1 \leqslant b \leqslant e} \left[L^*(b-1) \times L^+(b,e) \right] \tag{22}$$

This equation is very similar to equation (14). The recurrent scheme for retrieving the pronounced utterance is the same as explained in the section 4.

3. TWO-DIMENSIONAL SYSTOLIC ARRAYS FOR SPEECH RECOGNITION

The speech recognition algorithms that have been described both have the property to be very regular in the sense that the basic computation to be executed at point (i,j) depends only on results provided by computations at neighboring points. This property makes it possible to implement the algorithms on systolic arrays. As we shall see, various systolic structures are possible. The purpose of this section is to explain how a two-dimensional array can support these algorithms. The following presentation is based on [3] for the DTW algorithm, and on [6][8] for the probabilistic matching algorithm. The systolic array for the connected word recognition was described in [7].

The basic idea is to associate one processing element to the computation of each value $D(i,j)$ in the case of the DTW algorithm, or to the values $L(i,j)$ in the case of the probabilistic matching. Consider an array of processors denoted $P_{i,j}$ connected as indicated by Fig. 6 (a) : $P_{i,j}$ has three input ports denoted I_v (vertical input), I_d (diagonal input) and I_h (horizontal input), and two output ports O_v, O_d and O_h (respectively vertical, diagonal and horizontal output ports). I_v of $P_{i,j}$ is connected to O_v of $P_{i-1,j}$, I_d is connected to O_d of $P_{i-1,j-1}$ and I_h is connected to O_h of $P_{i,j-1}$. In order to clarify the explanation in the following, we assume that each processor has the functional architecture depicted by Fig. 6 (b).

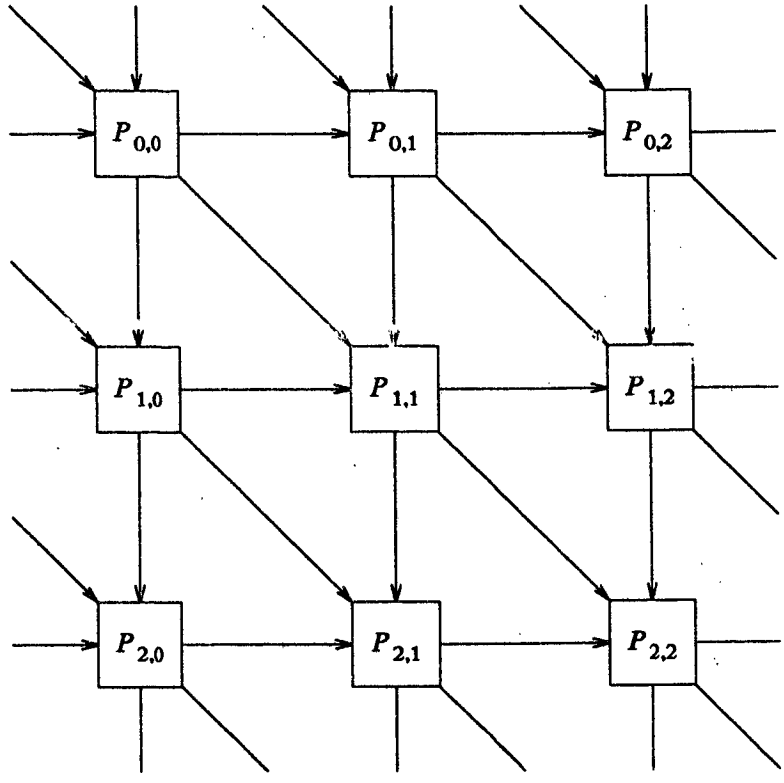

Fig 6a: Two-dimensional Systolic Array for Connected Speech Recognition.

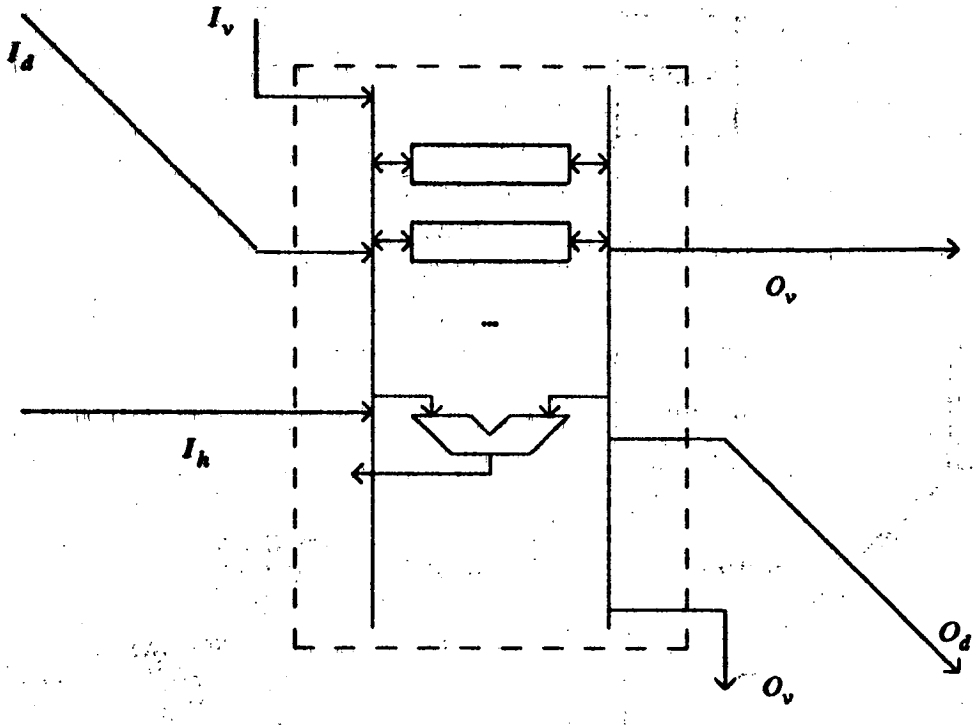

Fig. 6b: Functional structure of one cell of the array.

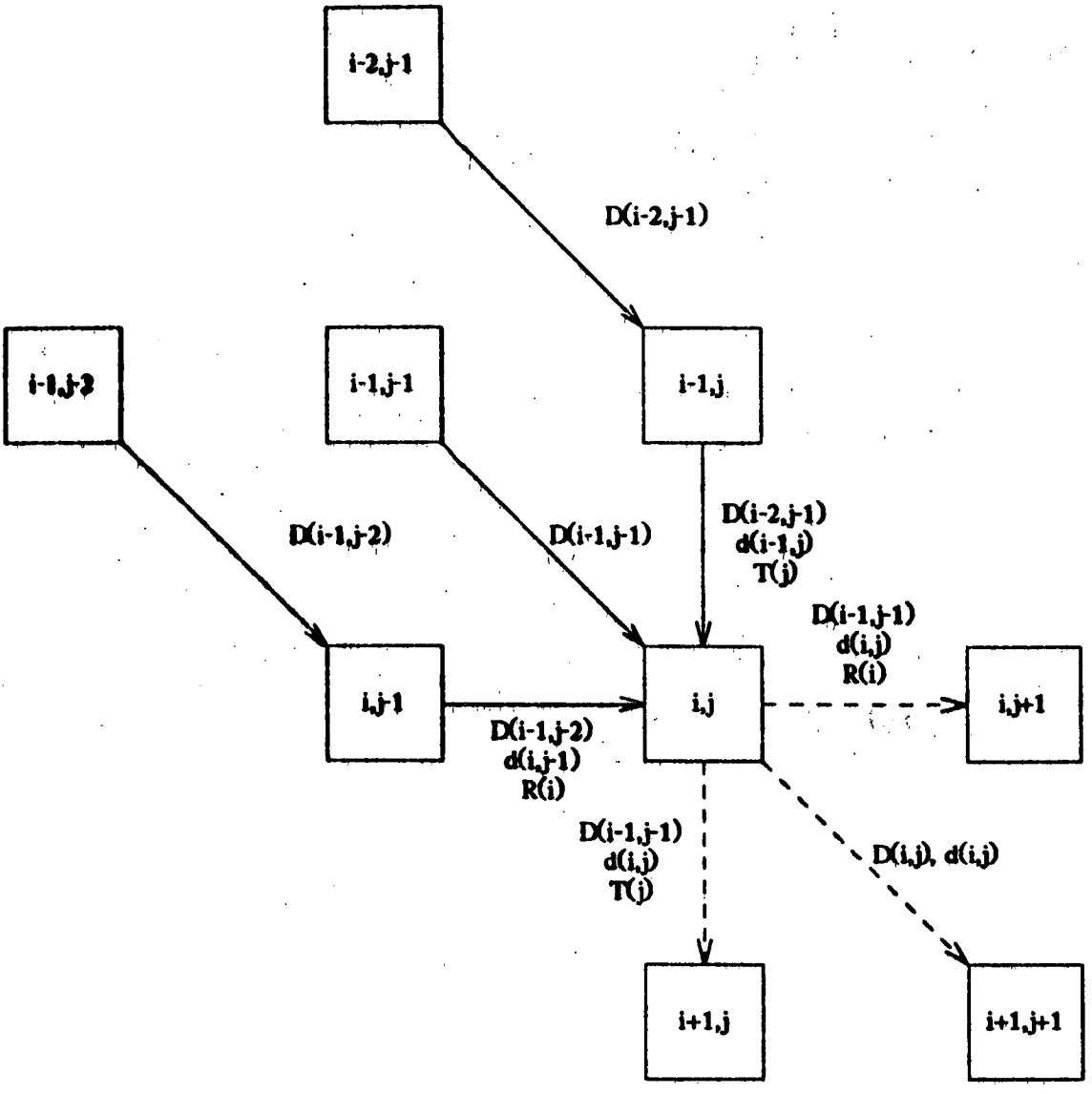

Fig. 7: Data transmission during a cycle performed by processor $P_{i,j}$ for the Probabilistic Matching Algorithm. Solid arrows show data that are received, and dashed arrows indicate data that are sent.

A set of registers memorize the partial calculations performed by the arithmetic unit. We assume that the I/O operations are performed synchronously on the whole array. We consider successively the case of the DTW algorithm and the probabilistic algorithm.

3.1. DTW-Algorithm

In the following section, we describe the elementary cell calculation, then the overall operation of the array, the pipelining mode of operation, and finally, how the array can be used for connected word recognition.

3.1.1. Elementary cell calculation

$P_{i,j}$ has to perform the computation expressed by equation (4), i.e:

$$D(i,j) = \min \begin{cases} D(i-1,j-2) + d(i,j-1) + d(i,j) \\ D(i-1,j-1) + d(i,j) \\ D(i-2,j-1) + \dfrac{d(i-1,j) + d(i,j)}{2} \end{cases} \tag{4}$$

Fig. 7 illustrates the way data have to be transmitted between the processors. Let us first focus on the data that are required by $P_{i,j}$ to compute (4). These data are represented by solid arrows on Fig. 7. $D(i-1,j-2)$ is produced by $P_{i-1,j-2}$, and has consequently to be routed to $P_{i,j-1}$ then to $P_{i,j}$. In the same way, $D(i-2,j-1)$ is routed through $P_{i-1,j}$ to reach $P_{i,j}$. Distance $D(i-1,j-1)$ may be routed through $P_{i,j-1}$, and then to $P_{i,j}$. The elementary distance $d(i,j)$ is computed by $P_{i,j}$, assuming $R(i)$ and $T(j)$ are known from $P_{i,j}$. But since $R(i)$ is used by all the processors of row i, $R(i)$ may be provided by $P_{i,j-1}$; similarly, $T(j)$ is provided by $P_{i-1,j}$. Finally, the elementary distances $d(i,j-1)$ and $d(i-1,j)$ are also provided by $P_{i,j-1}$ and $P_{i-1,j}$ respectively.

$P_{i,j}$ has not only to perform the computation of equation (4), but also to provide processors $P_{i,j+1}$, $P_{i+1,j+1}$ and $P_{i+1,j}$ with the data they need. These data are represented by dashed arrows on Fig.7. Processor $P_{i,j}$ sends $D(i,j)$ and $d(i,j)$ to processor $P_{i+1,j+1}$, and sends $D(i-1,j-1)$ and $d(i,j)$ to the processors $P_{i+1,j}$ and $P_{i,j+1}$. The program that must be performed by $P_{i,j}$ is given by Fig. 8. In the following, the duration of this program will be called the cycle of the systolic array.

3.1.2. Overall operation of the systolic array

The overall operation of the two-dimensional systolic array for the DTW algorithm is made on a diagonal basis. Consider the comparison of a reference pattern R and of a test pattern T. Assuming that the computation start at time 0, at time t, all the processors $P_{i,j}$ such that $i + j = t$ are active. In such a way, it can be easily checked that all the arguments needed for the computation of equation (4) have already been computed and have been routed correctly. The results $D(N,j)$ are obtained by the processors of the bottom row:

{ Program performed during a systolic cycle
$D, d, D_v, D_h, D_d, d_v, d_h, d_d, T$ and R
are registers of the elementary processor.
Instructions separated by | are executed
simultaneously.}

{ Read new value T(j) and send old one }

T <- I_v | T -> O_v

{ read D(i-2,j-1), and send old D(i,j) }

D_v <- I_v | D -> O_v

{ read d(i-1,j) and send old d(i,j) }

d_v <- I_v | d -> O_v

{ read D(i-1,j-1) and send old D(i,j) }

D_d <- I_d | D -> O_d

{ Read new value R(i) and send old one }

R <- I_h | R -> O_h

{ read D(i-1,j-2) and send old D(i,j) }

D_h <- I_h | D -> O_h

{ read d(i,j-1) and send old d(i,j) }

d_h <- I_h | d -> O_h

{ compute new d(i,j) }

$$d = ||R - T||$$

{ compute new D(i,j) }

$$D = \min \left(D_v + d_v + d \, , \, D_d + d \, , \, D_h + \frac{d_h + d}{2} \right)$$

Fig. 8: Program performed by a cell during a basic cycle.

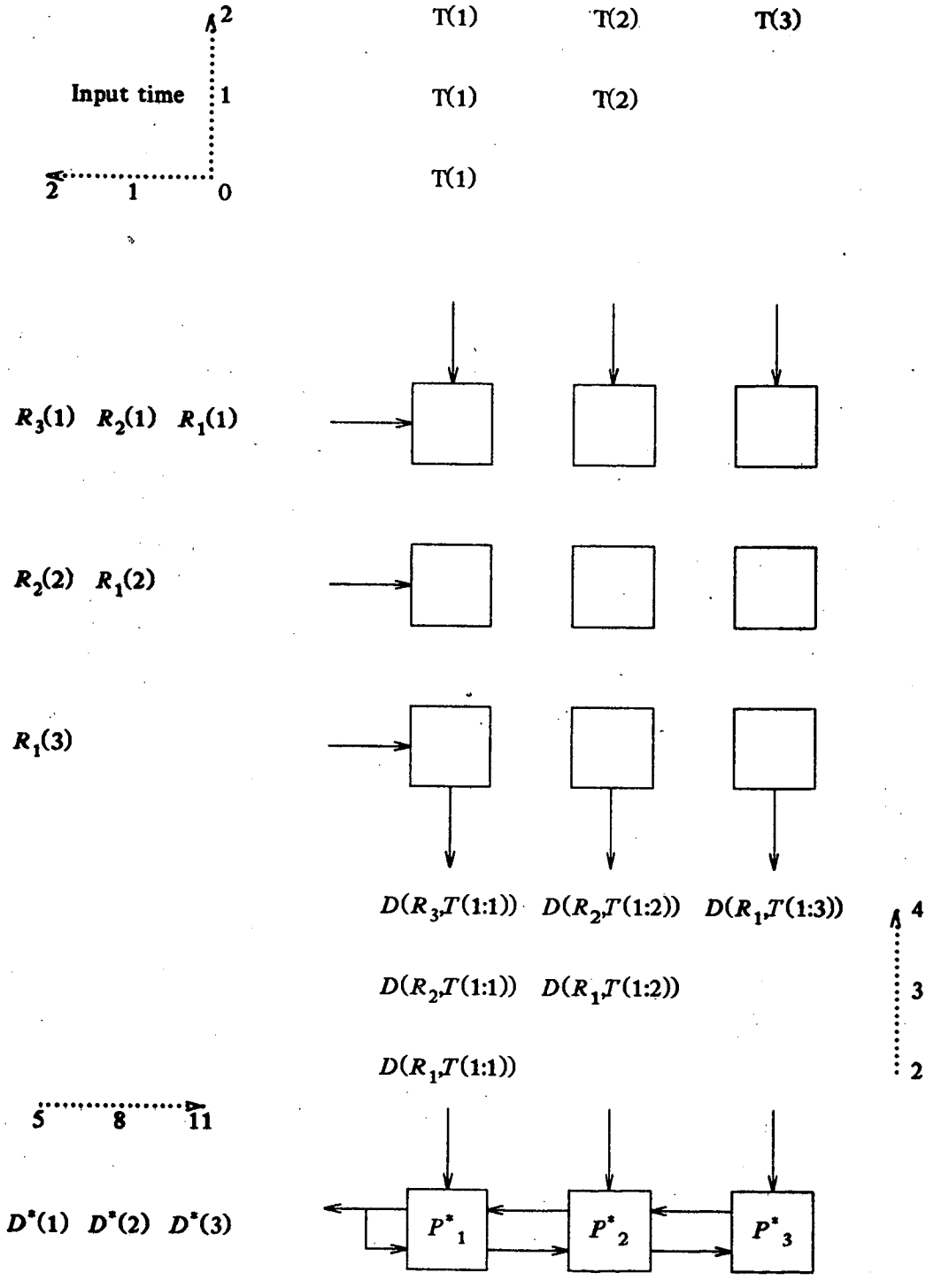

Fig. 9: Pipelining three reference words into the systolic array. The time at which the data enter or leave the array are indicated on dotted arrows.

$P_{N,j}$ delivers $D(N,j)$ at time $N + j - 1$.

In order to cope with variable length reference patterns, the array is dimensioned according to the longest reference word. Let $N_m = \max_{1 \leqslant v \leqslant V} N_v$ be the maximum number of frames of the reference words. The array has thus N_m rows and $N_m + r$ columns. The process remains essentially the same if the length N_v of the reference is smaller than N_m. In such a case, the final values $D(N_v,j)$ are computed by processors of row N_v, and must be transmitted to the bottom of the array. One way to solve this problem is to associate a flag with each reference frame $R(i)$. This flag is set to 1 for $i \geqslant N_v$ and set to 0 otherwise. A processor receiving this flag set to 1 from its left neighbor transmits the value $D(i,j)$ instead of $D(i-1,j)$ to its bottom neighbor, and $+\infty$ instead of $D(i,j)$ to $P(i+1,j+1)$. In such a way, the processors of the rows N_v, N_v+1, etc., propagate the final values $D(N_v,j)$ to the bottom boundary of the array.

3.1.3. Pipelining the reference words

Since only one diagonal of this network is active at a time for the computation of the DTW algorithm, several computations may be pipelined into the array as illustrated on Fig. 9. The pipeline operation of the array allows all the distances between any reference and any sub-pattern of T to be computed very efficiently. A sub-pattern $T(b : b+N_m+r-1)$ that enters the top of the array remains constant, and the successive reference patterns R_v, $1 \leqslant v \leqslant V$ enter the left part of the array. It can be seen that after a certain delay, each processor $P_{N_m,j}$ of the bottom row of this array will output values $D(R_v,T(b:b+j-1))$, for $1 \leqslant v \leqslant V$, one during each systolic cycle. When all the vocabulary has been matched against the test pattern, it remains to shift the test pattern one position to the left before repeating the process. In the following, we will refer to the process of comparing the whole vocabulary with a sub-pattern of the test utterance as a **vocabulary cycle.**

3.1.4. Connected speech recognition

The computation of $D^*(e)$ (defined by equation (14)) is carried out by a linear row of processors connected to the bottom row of the two-dimensional array, as depicted on Fig. 9. The processors of this extra row are denoted as P^*_j, $1 \leqslant j \leqslant N_m+r$. Recall that:

$$D^*(e) = \min_{1 \leqslant b \leqslant e} \left[D^*(b-1) + D^+(b,e) \right] \tag{14}$$

During a vocabulary cycle, a processor P^*_j receives the quantities $D(R_v,T(b:b+j-1))$, that are output by processors $P_{N_m,j}$ and can thus perform the minimization:

$$D^+(b,b+j-1) = \min_{1 \leqslant v \leqslant V} D(R_v,T(b:b+j-1))$$

$P^*_{m,j}$ also keeps track of the number $v^+(b:b+j-1)$ of the reference pattern which achieves the minimum. Assume for the moment that processor P^*_j receives from its left neighbor the quantity $D^*(b-1)$ (we shall see later on how this can be done). Then P^*_j is able to compute the quantity $D^*(b-1) + D^+(b,b+j-1)$. But since the test utterance is shifted one

position to the left at the end of the vocabulary cycle, processor P^*_{j-1} will receive during the next vocabulary cycle the quantities $D(R_v,T(b+1,b+j-1))$ and, in a similar way, will compute $D^+(b+1,b+j-1)$ and $D^*(b) + D^+(b+1,b+j-1)$. In order to achieve the minimization over b of equation (14), it is sufficient that P^*_j sends to P^*_{j-1} at the end of each vocabulary cycle the final value $D^*(b-1) + D^+(b,b+j-1)$ it has found. This value is compared to the value $D^*(b) + D^+(b+1,b+j-1)$ computed by P^*_{j-1} during the next vocabulary cycle. In such a way, each step of the minimization is carried out in turn by the processors of the bottom row, from the right to the left. Therefore, the final results $D^*(e)$ are computed by processor P^*_1. It remains now to explain how values $D^*(b)$ may reach the processors at the right time. Note that all the processors need the same value $D^*(b)$ at the end of a given vocabulary cycle. But this value has just been produced by P^*_1 at the end of the previous vocabulary cycle. Therefore, if this value reenters P^*_1 and move from left to right one processor every systolic cycle, it is broadcasted to all the cells of the bottom row.

The above implementation requires $(N_m+1)\times(N_m+r)$ processors, if the full array is implemented. If only processors $P_{i,j}$ such that $|i - j| \leqslant r$ are implemented, a simple calculation shows that $2(r+1)N_m - \frac{r(r-1)}{2}$ are required. Since the vocabulary cycles are pipelined, one reference word enters the array during each cycle. The number of systolic cycles needed to compare the whole vocabulary is thus $VM + 2N_m + r$, where M is the length of the test pattern. In other words, the array does V systolic cycles for each test frame.

3.2. PROBABILISTIC MATCHING WITH A TWO-DIMENSIONAL ARRAY

The overall operation of the array is similar to those of the DTW-algorithm. In particular, the pipelining scheme remains the same. In this section we will focus on the operation of the basic processor, and on the main problem in implementing this algorithm, namely the memory size requirement. Consider the comparison between a reference pattern R containing N phonemes (including the end of word marker]) and a test pattern T having M frames. We assume that the last test frame $T(M)$ is composed uniquely of the end-of-word marker], with probability 1. The array needed to implement such a computation has N rows and M columns. Processors are numbered $P_{i,j}$ where $0 \leqslant i \leqslant N-1$ and $0 \leqslant j \leqslant M-1$. Fig. 10 describes the computations performed by processor $P_{i,j}$. Processor $P_{i,j}$ receives $T(j+1)$ and $L_o(i,j)$ on its I_v port, $L_c(i,j)$ on I_d and finally $R(i+1)$ and $L_i(i,j)$ on I_h. Using equation (17), the processor computes then $L(i,j)$. Then using equations (18), (19) and (20), it computes $L_o(i+1,j)$ which is sent together with $T(j+1)$ on O_v, then $L_c(i+1,j+1)$ which is sent on O_d, and finally $L_i(i,j+1)$ which is sent together with $R(i+1)$ on O_h.

Note that this process still remains valid for processors at the boundaries of the array. As far as processors of the top row and the left column are concerned, equation (21) defines the initial values L_o, L_c and L_i that must enter these processors. Consider now processors $P_{N-1,j}$ for $j<M-1$. Since we assume that each reference word is terminated with a special marker], and since $P_c(]) = 0$, we can deduce from (20) that:

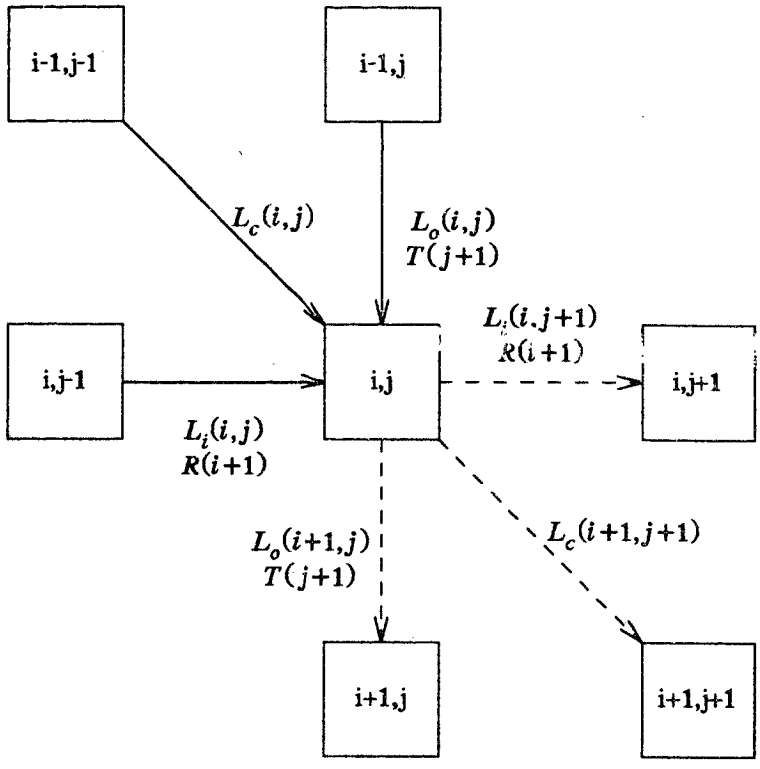

Fig. 10: Data transmission during a cycle performed by processor $P_{i,j}$ for the Probabilistic Matching Algorithm.

$$L(N,j) = L_o(N-1,j) \tag{23}$$

Therefore, the final value $L(N,j)$ is delivered by $P_{N,j}$ on its O_v port. Finally, consider the processors $P_{i,M-1}$ of the right column. Provided that we have $q_c(\square|y) = q_i(\square|y) = 0$ for all phoneme y, it can be seen that $P_{i,M-1}$ still delivers $L_o(i+1,M-1)$ on its vertical output port O_v. Therefore, from (23), $P_{N-1,M-1}$ provides $L(N,M-1)$ on its O_v port.

Note that this operation assumes that each processor knows the probabilities distributions P_o, P_c, P_i as well as q_o, q_c and q_i. However, the pipelining scheme described in the previous section helps to reduce significantly the amount of memory required, since the test pattern remains unchanged during a whole vocabulary cycle. As a consequence, processor $P_{i,j}$ receives the same frame $T(j+1)$ during V consecutive systolic cycles. This suggests loading the processors with the probabilities $q_c(x|y)$ and $q_i(x|y)$ only for the phonemes x which are in $T(j+1)$. Before every vocabulary cycle, these parameters are loaded into the first row of the array, then the second, etc., until the whole array is initialized.

4. CONNECTED SPEECH RECOGNITION USING A LINEAR SYSTOLIC ARRAY

The following section presents two differents schemes that may be used for the real-time connected speech recognition of vocabularies of up to a few hundred words.

The systolic structure that can support these algorithms is made out of processing elements numbered P_i, that are two-way linearly connected as depicted on Fig. 11. Each processing element has two input ports denoted as I_l (input from the left) and I_r (input from the right) and similarly two output ports, O_l and O_r. This structure can be used in a number of different ways to perform the computations of the DTW algorithm or the Probabilistic Matching. The basic idea is to make this linear array perform the computation of either successive rows, diagonals, or columns of the two-dimensional array previously described. For reasons that will be made clear in the following development, will shall here consider the diagonal scheme only for the DTW algorithm, and the row scheme for the probabilistic matching algorithm.

4.1. DTW USING A DIAGONAL SCHEME

Consider the comparison between a reference pattern R of N frames, and a test pattern T of M frames. We have to compute $D(i,j)$ for (i,j) such that $1 \leqslant i \leqslant N$, $1 \leqslant j \leqslant M$ and $|i-j| \leqslant r$. For the sake of simplicity, we assume $r = 2q$ to be even. We have seen in section 3.1 that a simple way to order the calculations is to have $D(i,j)$ computed at time $t(i,j) = i+j$. A first systolic implementation is obtained by assuming that P_k carries out all the computations $D(i,j)$ such that $i-j = k$. However, this implementation has the drawback that each processor is only working every other time. Another more interesting implementation consists in having P_k compute the values $D(i,j)$ such that $\left|\frac{i-j}{2}\right|$ (where $\lfloor x \rfloor$ denotes the greatest integer lower than or equal to x) as depicted on Fig. 12 (a). In such a way, the linear array has $r+1$ processors numbered P_{-q} through P_q.

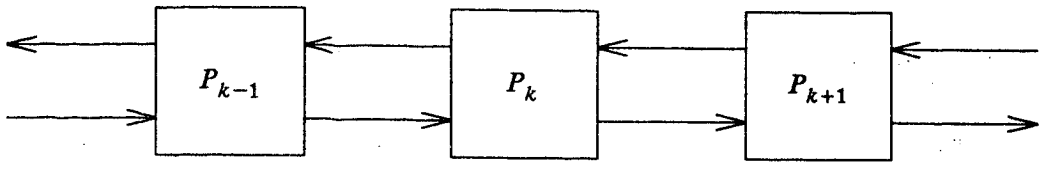

Fig 11 (a): Structure of the linear array.

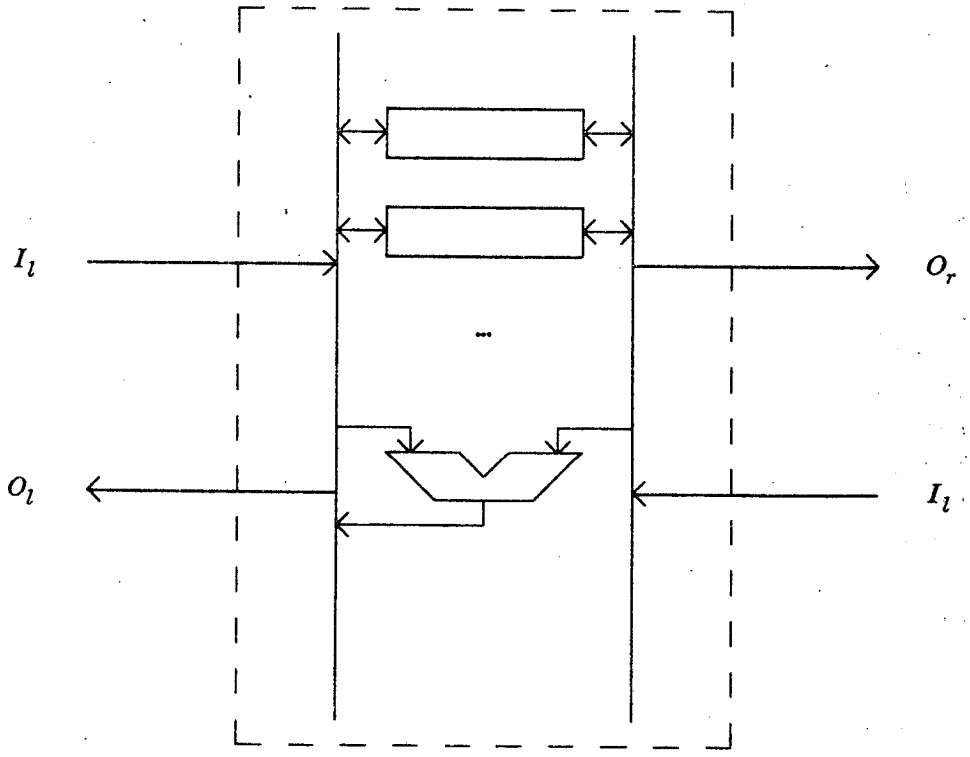

Fig. 11 (b): Functional structure of the cells.

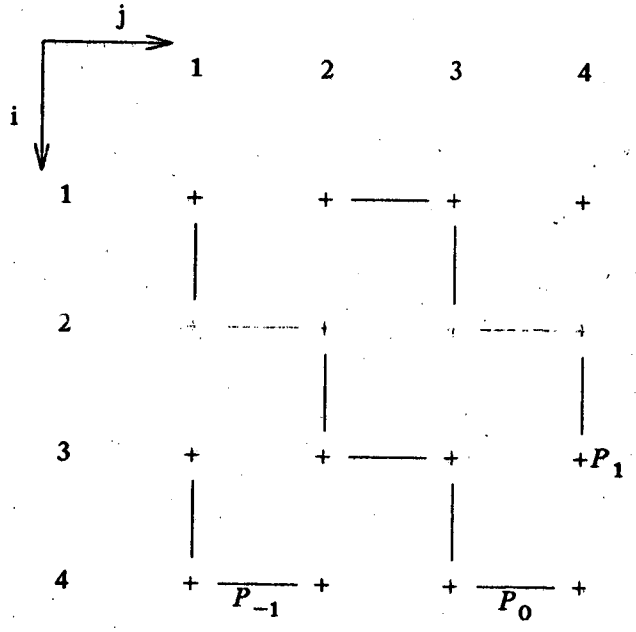

Fig. 12 (a): Trace of the computations executed by the processors during the execution of the Dynamic Time Warping Algorithm on the linear systolic array. Solid lines join the points corresponding to the calculations performed by each processor.

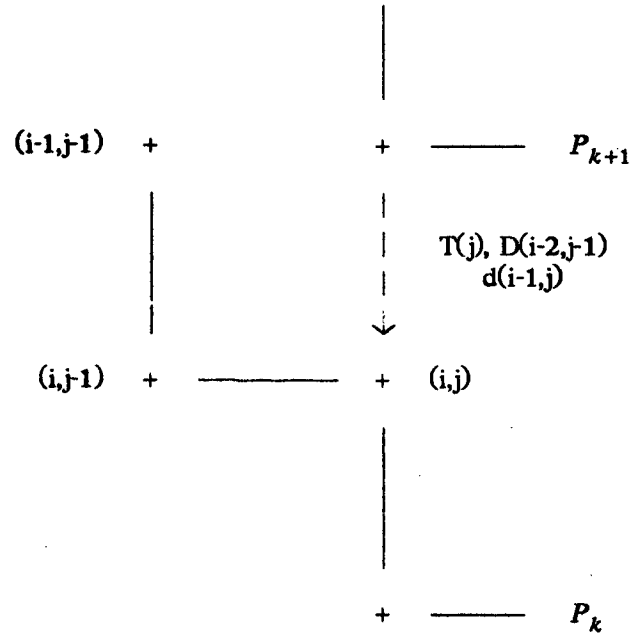

Fig. 12 (b): Data transfers during a T-cycle. Solid lines join the calculations performed on the same processor. Dashed arrows show data transfers.

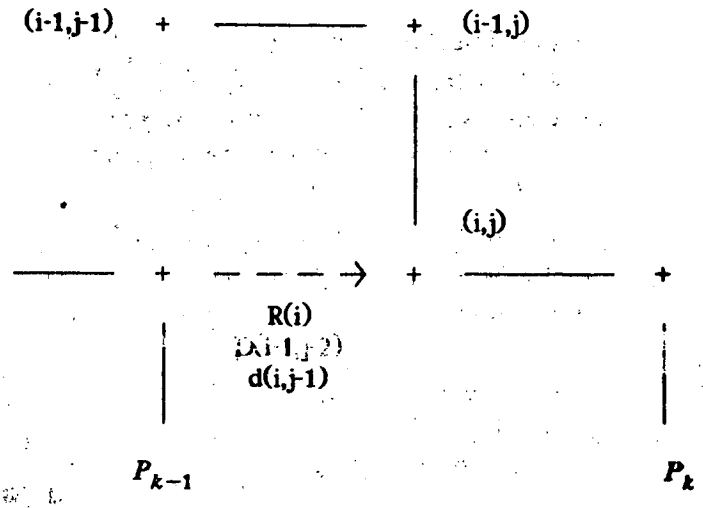

Fig. 12 (c): Data transfers during a R-cycle.

{ Program performed during a T-cycle
$D, d, D_v, D_h, D_d, d_v, d_h, d_d, T$ and R
are registers of the elementary processor }
{ Read new value T(j) and send old one }
T <- I_r | T -> O_l
{ read D(i-2,j-1), and send old D(i,j) }
D_v <- I_r | D -> O_l
{ read d(i-1,j) and send old d(i,j) }
d_v <- I_r | d_h -> O_l
{ compute new d(i,j) }
$d = ||R - T||$
{ compute new D(i,j) }
$$D = \min\left(D_h + d_h + d\ ,\ D_d + d\ ,\ D_v + \frac{d_v + d}{2}\right)$$
{ Prepare nest R-cycle }
D_v <- D_d
D_d <- D_p
D_p <- D
d_v <- d

Fig. 13 (a): Program performed by a cell during a T-cycle.

{ Program performed during a R-cycle }
{ Read new value R(j) and send old one }
R <- I_l | R -> O_r
{ read D(i-1,j-2), and send old D(i,j) }
D_h <- I_l | D -> O_r
{ read d(i,j-1) and send old d(i,j) }
d_v <- I_l | d_h -> O_r
{ compute new d(i,j) }
$d = ||R - T||$
{ compute new D(i,j) }
$$D = \min\left| D_h + d_h + d \;,\; D_d + d \;,\; D_v + \frac{d_v + d}{2} \right|$$
{ Prepare nest T-cycle }
D_h <- D_d
D_d <- D_p
D_p <- D
d_h <- d

Fig. 13 (b): Program performed by a cell during a R-cycle.

4.1.1. Permanent regime

Let us first examine the operation of each processor when permanent regime is attained. P_k executes two different cycles depending on whether $i-j = 2k$ or $i-j = 2k+1$.

Case 1: $i-j = 2k$ **(Fig. 12 (b))**

In this situation, P_k has already the values $R(i)$, $d(i,j-1)$ and $D(i-1,j-2)$ that were needed for the calculation of $D(i,j-1)$ during the previous cycle. It contains also the value $D(i-1,j-1)$ which was used two cycles before. Values $D(i-2,j-1)$, $d(i-1,j)$ and $T(j)$ are provided by processor P_{k-1}. Well shall refer to this cycle as a T-cycle (for T transmit cycle) later on. Fig. 13 (a) give a more detailed description of this cycle.

Case 2: $i-j = 2k+1$ **(Fig. 12 (c))**

Symmetrically, P_k contains already values $T(i)$, $d(i-1,j)$ and $D(i-2,j-1)$ obtained during the previous cycle, and $D(i-1,j-1)$ obtained two cycles before. It remains to get $D(i-1,j-2)$, $d(i,j-1)$ and $R(i)$ from processor P_{k+1}. This cycle is called a R-cycle (for R transmit cycle, see Fig. 13 (b)).

4.1.2. Initialization and termination

The initialization and termination process need to be examined carefully, since it is very important to keep the overall process regular: in particular, it is most important that data enter or leave the array only through the extremal processing elements, namely P_{-q} and P_q, in such a way that the number of connections with the other parts of the system are minimized.

During the initialization cycles, data enter the array in such a way that all the processing elements reach a consistent state and thus become able to perform their first computation. This is achieved by performing particular initialization cycles referred to as TI-cycle and RI-cycle. During a TI-cycle, processor P_k reads a test frame $T(j)$ from P_{k+1}, and initialization values for the registers D_v, d_v, D_d and D_p. During a RI-cycle the processor P_k reads a reference frame $R(i)$ and initialization values for the registers D_h and d_h. The initialization sequence consists of $q-1$ TI-cycles and q RI-cycles.

The termination process is carried out by making the final values $D(N,j)$ move to processor P_{-q} so that they may be output. After a processor has received the last reference frame $R(N)$, the processor does not compute value $D(i,j)$ which is not significant any more, but instead propagates during the T-cycles the value $D(i,j)$ it receives from its right neighbor. This termination scheme has the advantage that the data are output only by P_{-1} and also that no extra hardware is needed to send the results. However, since the last result $D(N_v,N_v+r)$ is produced by P_q, $2r$ cycles are necessary before this result reaches P_{-q}, introducing a significant overhead. Another way to proceed is to have each processor access a common output bus so that the results are sent directly to the outside. This is possible, since at a given time, at most one processor produces a final result.

The number of processors required to implement this scheme is $r+1$. The comparison between a reference R_V and a pattern T consists in $r-1$ initialization cycles and $2N_v+r-1$ calculation cycles, assuming that the results are output directly by each processor. Therefore,

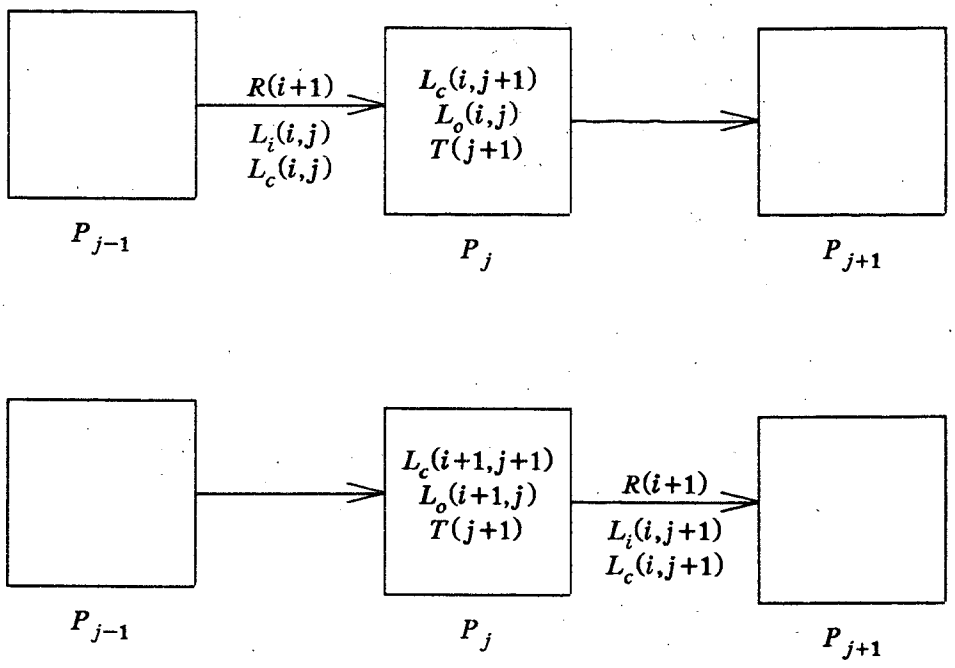

Fig 14: Data movement during a basic cycle of processor P_j (a) before computation of $L(i,j)$ and (b) after.

the total number of systolic cycles for one comparison is $2(N_v+r)$. This means that the array has $V \times 2(N_v+r)$ systolic cycles to perform for each test frame.

4.2. PROBALISTIC MATCHING USING A ROW SCHEME

Although it is possible to use the diagonal scheme to implement the probabilistic matching algorithm, this scheme has the main drawback that the probability values q_c and q_i have to be left-shifted every cycle, since the test frames flow from right to left.

A better scheme is to have processor P_j compute values $L(i,j)$, $(i = 1, \cdots, N)$, in such a way that frame $T(j)$ stays in P_j during a whole vocabulary cycle, thus minimizing data transfers between processors. In this implementation, called the **row scheme**, the linear array emulates successives rows of the two-dimensional array described in section 3. Therefore, the linear array has M processors numbered P_0 through P_{M-1}.

To explain in more details the operation of the array, consider the comparison between a reference pattern of N frames and a test pattern T; assume that processor P_j is already loaded with $T(j+1)$ and the probabilities $q_c(x|y)$ and $q_i(x|y)$ for $x \in T(j+1)$. During a basic cycle, P_j has to compute $L(i,j)$ according to equations (17), (18), (19) and (20) (see Fig. 14). $L_o(i,j)$, which has been computed by P_j during the previous cycle is already in P_j. Values $L_c(i,j)$, $R(i+1)$ and $L_i(i,j)$ are obtained from P_{j-1}. Processor P_j computes then $L(i,j)$, $L_i(i,j+1)$, $L_c(i+1,j+1)$ and $L_o(i+1,j)$. Values $L_i(i,j+1)$ and $R(i+1)$ are then sent to P_{j+1}. The probability $L_c(i+1,j+1)$ has to stay in processor P_j, since P_{j+1} will need it only two cycles later. Instead, $L_c(i,j+1)$ which was kept from the previous cycle is sent to P_{j+1}.

If we assume that the computation of $L(0,0)$ is done at time 0, then processor P_j computes $L(i,j)$ at time $i+j$. Final results $L(N,j)$ $(0 \leqslant j \leqslant M)$, are thus obtained respectively by $P_0, \cdots, P_j$ at time $N+j$.

In the context of connected speech recognition, these results need not to be sent outside the array. During a vocabulary cycle, processor P_j computes successively the probabilities $L(N,j)$ for all the references words of the vocabulary. P_j can therefore compute the quantity $L^+(b,b+j)$ by keeping track of the maximum probability over the whole vocabulary. At the end of the vocabulary cycle, a special cycle is required to carry out the minimization of equation (22).

The row scheme requires N_m+r+1 processors, assuming that N_m is the maximum number of phonemes of the reference words (final marker excluded). The number of systolic cycles for a single comparison is $2(N_v+1) + r$. Note however that the vocabulary cycles may be overlapped, since as soon as a processor P_j ends a vocabulary cycle, it can start the next one. In such a way, N_v+1 cycles are in fact needed for each comparison. Therefore, the array does $V \times (\bar{N}+1)$ cycles for each test frame, where $\bar{N}$ denotes the average number of phonemes of the reference words.

5. THE API89 CHIP

The following section is devoted to the structure of a special-purpose chip named API89 which implements one basic processor of the various systolic organizations that have been presented [9]. The design of this chip was made having in mind the following goals:

- try to have the maximum speed by using special-purpose elements required by the algorithms that must be supported by the chip;
- reduce the space layout by implementing only the functions that are needed;
- keep the processor general enough to support the different variations that are commonly used in the class of speech recognition methods here above presented.
- finally, choose a structure simple enough to avoid difficulties in designing and testing the chip.

5.1. Overall organization and Control

These ideas lead to the organization depicted by Fig. 15. The various elements of the circuit are organized around a single bus. These elements are:

a) Two input registers denoted as VR (vertical register) and HR (horizontal register);

b) One output register OR;

c) An Arithmetic Unit (AU) capable of performing addition, subtraction and incrementation on 16-bit values; the AU has an accumulator (ACC) and an input register (AR);

d) An array of 16 general purpose registers R[0] to R[15];

e) A 60 12-bit word memory with a Memory Address Register (MAR), a Memory Read Register (MR) and a Memory Write Register (MW);

f) A look-up table called the Z module, implemented using a PLA, for the summation of logarithms; this look-up table has an input register IZ and an output register OZ.

All the registers and the data paths are 16-bit wide in order to provide enough precision for the calculations as well as fast I/O. Operation of the processor is entirely synchronous and based on a two-phase nonoverlapping clock scheme. These two phases are denoted as ϕ_1 and ϕ_2 in the following.

The operation of the circuit is controlled by 10-bit micro-instructions which are generated outside the chip and expanded inside using a very simple control unit. Two types of micro-instructions have been defined as displayed on table 16. Microinstructions of the first type are executed during ϕ_1, and concern the transfers of data between the different modules through the bus. Microinstructions of the second type are synchronized on ϕ_2 and are used for I/O, memory and arithmetic operations. In order to increase speed, instructions are latched in a pipeline register thus allowing instruction decoding and execution to overlap. One instruction is thus executed every clock phase.

5.2. Description of the modules

Three modules of the chip need to be explained in more detail.

5.2.1. The memory

The memory is organized in such a way that the parameters necessary for the probabilistic matching algorithm can be memorized. We have seen that a processor need to keep track of the probabilities $q_c(x|y)$ and $q_i(x|y)$ for only three candidate phonemes. However, since there are about thirty-five phonemes in a language as English or French, the amount of

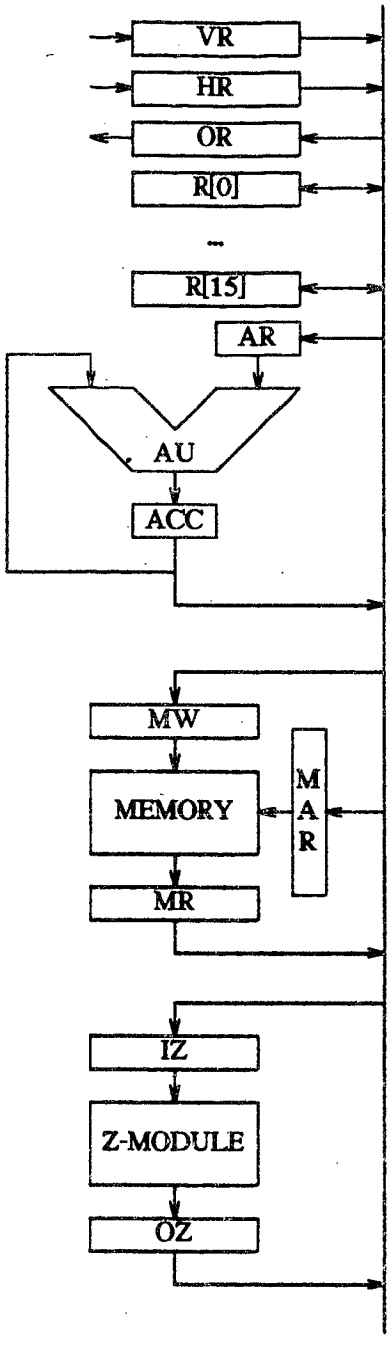

Fig. 15: Architecture of the API89 chip.

ϕ1 Instructions:	register transfers	
	Origin	Destination
	R[ ]	R[ ]
	ACC	AR
	OZ	IZ
	MW	MR
	HR	OR
	VR	MAR
ϕ2 Instructions	RM	Read Memory
	WM	Write Memory
	RR	Read for Refresh
	LDI	Load Input Register
	RST	Reset Accumulator
	INC	Increment Accumulator
	ADD	Add to Accumulator
	SUB	Substract from Accumulator
	STR	Conditionally Substract from Accumulator
	SBS	Conditionally Complement Accumulator

Table 16: API89 instruction set.

memory needed is too large to be reasonably implemented using the currently available technology. Two simplifications of the model are made according to an earlier software implementation of the algorithm [17]. First of all, it is assumed that the probability to insert a phoneme x is independent of the phonetic symbol y. Secondly, the probability to confuse a vowel and a consonant is assumed to be null. Therefore, for a given candidate phoneme x, it is only needed to memorize the quantities $q_i(x|y)$ and $q_c(x|y)$ for phonemes y which are of the same type (consonant or vowel) as x. Since we consider only three candidate phoneme for each test frame, the memory contains only 60 12-bits words, each candidate phoneme being represented on 20 words (this value has been chosen since there are 20 consonants in French). Each 12-bit word has two fields: a flag indicating the type of the phoneme, and a probability value coded on 11 bits.

The instructions for the memory are given by table 16. The RM instruction has a particular effect: if the type of the phoneme which has been loaded into the memory address register MAR and the type of the phoneme memorized in the memory word addressed are identical, then the value contained in that memory is loaded into the memory read register MR. Otherwise, the number representing probability 0 is loaded. In such a way, only one cycle is required to read a value $q_c(x|y)$ whatever the type of x and y are. Since the memory is implemented using dynamic memory cells, a particular instruction called RR is provided for memory refreshment.

5.2.2. Arithmetic Unit

The arithmetic Unit can perform addition, subtraction and incrementation as indicated on table 16. Two conditional instructions denoted as STR (star) and SBS (substar) are provided: STR complements the accumulator if the AU flag is set to 1; SBS subtract the AU input register to the ACC if the AU flag is set to 1. The AU flag is set to 1 when an arithmetic operation results in a negative number. As a consequence, the AU is capable to emulate comparison and absolute value instructions.

5.2.3. The Z module

Since the probabilistic matching algorithm has to perform multiplications, and manipulate very small values, the probabilities are coded using radix-2 logarithms. However, examining formulas (18), (19), and (20) reveals that we have to perform the calculation of $Log(a+b)$ given $Log\ a$ and $Log\ b$. This problem has been solved in the following way. Let Z be the real mapping defined by:

$$Z(t) = Log(1 + 2^t)$$

We have then:

$$Log\ (a+b) = Log(a) + Z(Log\ b - Log\ a) = Log(b)\ Z(Log\ a - Log\ b)$$

Given the value of the function Z on the interval $[\ 0\ ,\ +\infty\ ]$, it is possible to compute $Log(a+b)$ from $Log(a)$ and $Log(b)$. Since the values are coded on 16 bits, it has been possible to implement the function Z using a PLA having only 128 product terms, with a good accuracy.

5.3. VLSI design of the chip

Two sets of chips have been fabricated. The first one using 5μ-NMOS technology was fabricated by the French MPC. The chip contains approximately 12,000 transistors and measures 5mm by 6mm. The second set was fabricated by MOSIS using 4μ-NMOS technology. The chip is housed in a 64-pin package since 16-bit wide parallel ports are used. The pin out is the following:

- 32 pins for the two input ports;
- 16 pins for the output port;
- 10 pins for the instructions;
- 4 pins for power and clocks.

Since the processor is microprogrammable, and since the inside decoding is reduced, access to the internal elements is relatively easy. A micro assembler has been written to compile tests programs and generate bit patterns for the circuit. Moreover, this micro-assembler is used for simulating the processor and thus generate a print out of the bit patterns expected on the output pads. The basic clock cycle has been estimated to be 500 ns for the 5μ version. The chip is currently being tested.

5.4. Example of systolic array implementation using API89

As an example, we show in this section how the chip can be used to implement the two-dimensional array for the probabilistic matching algorithm. Fig. 17 depicts the interconnection of the array, for $N = M = 3$. We assume, moreover, that the full array is implemented. The operation of the array is fully synchronous. A control unit (CU) broadcasts the microinstructions to all the processors of the array, and provides a separate flow of microstructions for the connected word recognition array. Control .lines are indicated by dashed arrows in the figure. Note that although each processor has only two input ports and one output port, all the logical connections needed can be emulated on this network, since the operation are synchronous. For example, vertical transmissions of data between processors are done by having all the processors load the output register with the value to be transmitted, and then read during the next cycle on the vertical input register. On the other hand, the left-to-right interconnection between the processors of the bottom row (see Fig. 9) is needed only for the broadcasting of the values $L^*(b)$ to compute equation (22). This can be emulated by having these values enter the last row of the array, move from left-to-right on this row, and be sent to the bottom processors.

A last remark that should be made concerns the synchronization of the connected word recognition row. Since the references are pipelined in the array, the operation of processor P^*_{j+1} is the same as the operation of P^*_j but occurs one cycle later. However, since each processor look for the maximum of the results $L(N,j)$ sent by the last row of the array, it is possible to align in time the connected word recognition array by having the network deliver values 0 when no significant comparison is done.

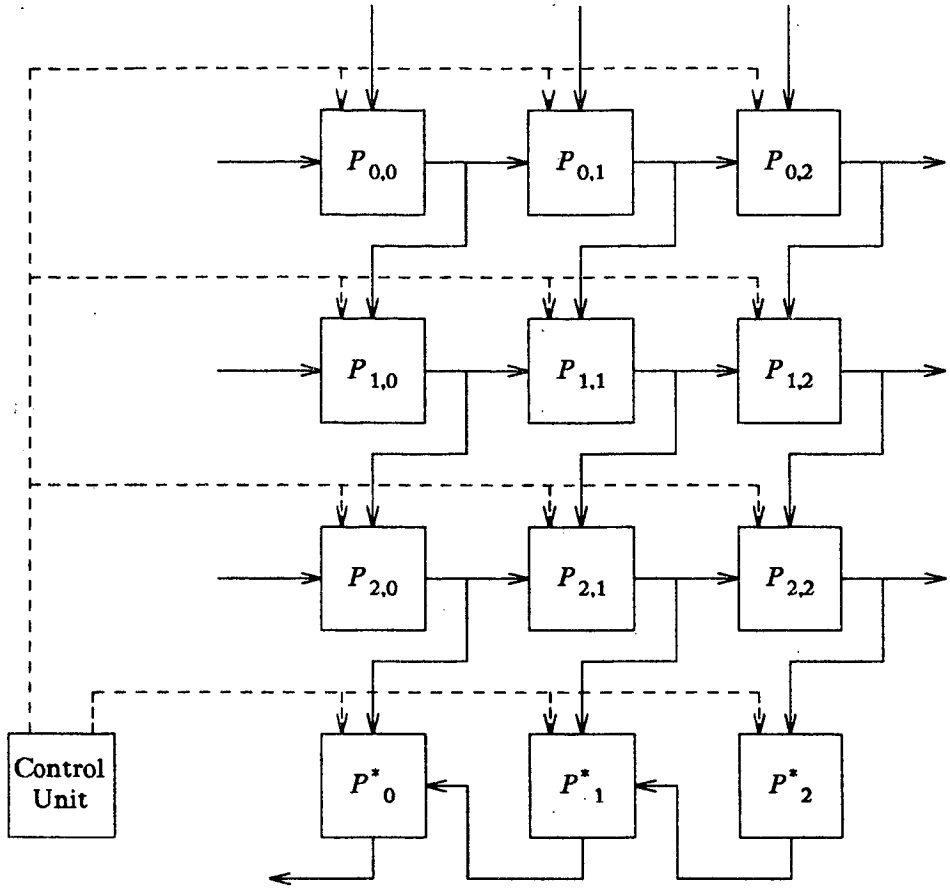

Fig. 17: Implementation of the two-dimensional systolic array for the probabilistic matching using API89.

6. DISCUSSION

Table 18 summarizes the characteristics of the four systolic arrays presented. For each implementation, an example of estimated size and performances is given, based on realistic parameter values.

The two-dimensional systolic array for the DTW algorithm leads to a 2,000 processor network when the full array is implemented, and to 925 processors when only the band of processors $P_{i,j}$ whith $|i - j| \leqslant r$ is implemented. This estimation is based on $N_m = 40$ and $r = 10$. Such an device would permit real-time recognition for a vocabulary of 5,000 words, based on a systolic cycle time of 4 microseconds, and a feature extraction interval of 20 milliseconds. This implementation is rather unrealistic for practical purpose due to the number of processors. It could only be considered if the number of processors were reduced by folding the computations. However, the control of this array would then be much more complicated.

The probabilistic matching algorithm may be run on a two-dimensional array having 180 processors if the full array is implemented, and 104 processors if the band is implemented. This estimation is based on $N_m = 10$ and $r = 4$. Executed on the API89 chip, the systolic cycle time is s=50 microseconds. Based on this parameter and assuming an average phoneme duration of 100 milliseconds, vocabularies of up to 2000 words could be handled in real time. Note however that the cycle time could be significantly reduced if the architecture of the chip was fully dedicated to the algorithm. The probabilistic matching implementation would be a good candidate for wafer-scale integration as presented in [5].

The linear systolic arrays are more realistic. The DTW algorithm implementation would permit a real time recognition of approximately 70 words, using 11 processors. On the other hand, 15 processors would allow the recognition of 250 words using the probabilistic matching algorithm. The difference of performances between these two systolic implementations comes from the fact that the initialization overhead is very significant in the case of the diagonal scheme. Although the row scheme could also be considered for the DTW algorithm, the number of processors would then be much higher, since N_m+r processors are needed. Both systolic arrays could be implemented on a single chip with the current available technology.

7. CONCLUSION

We have described several systolic arrays architectures which can implement two basic connected speech recognition methods. The API89 VLSI chip has been presented and it has been shown how such a chip can be used as the basic processor of these architectures. This study shows that there are many different ways to implement in parallel tasks such as connected speech recognition, even if one restricts oneself to a single type of architectures. Choosing between the different possibilities implies investigating in great details how the algorithm may be implemented, taking into consideration parameters such as the number of processors, the communications between the processors, and the complexity of the control. Our belief is that the new avenues opened by VLSI technology for special purpose hardware will become more and more practical, provided more is known about the various ways to

Method	Number of Processors	Number of cycles per Test frame	Example
Two-dimensional DTW Full array	$(N_m+1)(N_m+r)$	V	$r=10$, $N_m=40$, $s=4\mu s$. 2000 proc., $V=5000$
Two-dimensional DTW Band array	$2(r+1)N_m - r(r-1)/2$	V	$r=10$, $N_m=40$, $s=4\mu s$. 925 proc., $V=5000$
Two-dimensional Prob. Matching Full array	$(N_m+2)(N_m+r+1)$	V	$r=4$, $N_m=10$, $s=50\mu s$. 180 proc., $V=2000$
Two-dimensional Prob. Matching Band array	$2(r+1)(N_n+1) - r(r-1)/2$	V	$r=4$, $N_m=10$, $s=50\mu s$. 104 proc., $V=2000$
Linear, diagonal DTW	$r+1$	$2V\times(\bar{N}+r)$	$r=10$, $\bar{N}=30$, $s=4\mu s$. 11 proc., $V=70$
Linear, row Prob. Matching	N_m+r+1	$V\times(\bar{N}+1)$	$r=4$, $\bar{N}=8$, $s=50\mu s$. 15 processors, $V=250$

Table 18: Comparison of the systolic implementations. N_m is the maximum number of frames in the reference words, $\bar{N}$ the average number of frames in the reference words, V is the vocabulary size and s is the systolic cycle duration. The feature interval time for the DTW algorithm is assumed to be 20 milliseconds, and the duration of a phoneme is assumed to be 100 milliseconds.

map an algorithm onto a parallel architecture.

ACKNOWLEDGMENTS

The authors are greatly indebted to the department of Computer Science of the University of Santa-Clara where Patrice Frison designed the API89 chip. The fabrication by MOSIS of one set of prototype chips has been made possible thanks to the department of Computer Science and the department of Electrical Engineering of the North-Carolina State University. The authors would like also to thank F. Anceau who was responsible of the French MPC.

REFERENCES

[1] L.J. Siegel, "Highly Parallel Architectures and Algorithms for Speech Analysis," *1984 Int. Conf. Acoustic., Speech, Signal Processing*, San Diego, March 1984, pp. 25A.1.1-25A.1.4 .

[2] H.T. Kung, "Why Systolic Architectures ?," *Computer*, Vol. 15, No 1, Jan 1982, pp. 37-46.

[3] N. Weste, D.J. Burr, and B.D. Ackland, "Dynamic Time Warp Pattern Matching Using an Integrated Multiprocessing Array," *IEEE Trans. Comp.*, Vol. C-32, Aug. 1983, pp. 731-744.

[4] M.A. Yoder and L.J. Siegel, "Dynamic Time Warping Algorithms for SIMD Machines and VLSI Processor Arrays," *1982 Int. Conf. Acoustic., Speech, Signal Processing*, Paris, May 1982, pp. 1274-1277.

[5] J.A. Feldman, S.L. Gaverick, F.M. Rhodes, and J.R. Mann, "A Wafer Scale Integration Systolic Processor for Connected Word Recognition," *1984 Int. Conf. Acoustic., Speech, Signal Processing*, San Diego, March 1984, pp. 25B.4.1-25B.4.4 .

[6] J.P Banatre, P. Frison, and P. Quinton, "A Network for the Detection of Words in Continuous Speech," *Proceedings of the VLSI 81 Int. Conference*, Edimburg, Aug. 1981.

[7] J.P. Banatre, P. Frison, and P. Quinton, "A Systolic Algorithm for Speech Recognition," *1982 Int. Conf. Acoustic., Speech, Signal Processing*, Paris, May 1982, pp. 1243-1245.

[8] J.P. Banatre, P. Frison, and P. Quinton, "A Network for the Detection of Words in Continuous Speech," *Acta Informatica 18, pp. 431-448.*

[9] P. Frison, "Un Processeur Integre pour la Reconnaissance de la Parole," INRIA Research Report No. 215, July 1983.

[10] L. R. Bahl, and F. Jelinek, "Decoding for Channels with Insertions, Deletions, and

Substitutions with Applications to Speech Recognition," *IEEE Trans. Informat. Theory*, Vol IT-21, No 4, pp 404-411, 1976

[11] R. Bellman, S. Dreyfus, *Applied Dynamic Programming*, Princeton University Press, New-Jersey, 1962.

[12] H. Sakoe and S. Chiba, "Dynamic Programming Algorithm Optimization For Spoken Word Recognition," *IEEE Trans. Acoust., Speech, Signal Processing*, Vol. ASSP-26, No. 1, Feb. 1978, pp. 43-49.

[13] J. Bridle, M. Brown, and R. Chamberlain, "An algorithm for Connected Word Recognition," *1982 Int. Conf. Acoustic., Speech, Signal Processing*, Paris, May 1982, pp. 899-902.

[14] C. Myers, and L.R. Rabiner, "A Level Building Dynamic Time Warping Algorithm for Connected Word Recognition," *IEEE Trans. on Acoust., Speech, Signal Processing*, Vol ASSP-29, No. 3, April 1083, pp. 351-363.

[15] H. Sakoe, "Two-Level DP-Matching - A Dynamic Programming-Based Pattern Matching Algorithm for Connected Word Recognition," *IEEE Trans. Acoust., Speech, Signal Processing*, Vol. ASSP-28, No. 4, Aug. 1980, pp. 588-595.

[16] G. Mercier, A. Nouhen, P. Quinton, and J. Siroux, "The Keal Speech Understanding System," In: Spoken Language Generation and Understanding, J.C. Simon (Ed), Proc. NATO ASI, Bonas, 1980.

[17] L. Buisson, G. Mercier, J.Y. Gresser, M. Querre, and R. Vives, "Phonetic Decoding for Automatic Recognition of Words", *Speech Communication Seminar*, Stockholm, Aug. 1974

Imprimé en France

par

l'Institut National de Recherche en Informatique et en Automatique