PARALLEL PARTITIONING TECHNIQUES FOR THE DTW ALGORITHM IN SPEECH RECOGNITION

by G.STAINHAUER AND G.CARAYANNIS

ABSTRACT

Two parallel techniques for the partitioned parallel implementation of a DTW algorithm are discussed in this paper. The parallel architectures proposed, are independent of the number of processors available, if they do not exceed a specific number related to the DTW strategy. The first technique leads to a circular array, while the second to a linear array of processing elements (PE). The advantage of both architectures is that their efficiency remains almost unchanged when the number of PEs varies.

I. INTRODUCTION

One of the most successful pattern comparison methods, which is used in isolated and connected speech recognition is the dynamic time warping (DTW). Recently, parallel techniques [2,3] have been proposed for its implementation, in order to achieve real time speech recognition taking advantage of VLSI technology.

Here, some new parallel schemes are presented, which can be implemented on multiprocessor systems. The first parallel structure leads to the design of a circular array while the second to a linearly connected array.

Among the various DTW algorithms, we have chosen the one proposed by H.Sakoe and S.Chiba [1] and denoted as the symmetric form with the slope constraint condition P=1. This version has been studied in many isolated word recognition systems. The basic DTW algorithm used in this paper is depicted in fig. 1, where speech patterns A (reference) and B (test) are developed along the i-axis and j-axis respectively. Patterns A and B are expressed as a time series of feature vectors $a_i$, $i=1, \ldots, I$ and $b_j$, $j=1, \ldots, J$ respectively, where $a=(a_0, a_1, \ldots, a_I)$ and $b=(b_0, b_1, \ldots, b_J)$. DTW is normally performed by using a dynamic programming equation (DPE). In our case the DPE is given by

\[
S(i,j) = \min \{ S(i-1,j-1)+2d(i,j), S(i-1,j)+d(i,j), S(i,j-1)+d(i,j) \} \tag{1.0}
\]

where $S(i,j)$ represents the partial sum to point $(i,j)$ along the best path from point $(1,1)$ to point $(i,j)$ and $d(i,j)=d(a_i,b_j)=|a_i-b_j|^2$ is the local distance at point $(i,j)$. The result of a DTW algorithm is affected by several factors including boundary conditions, adjustment window conditions etc. For definitions and discussions of these factors see [1]. The architectures introduced here, can also be used for the implementation of some more DTW versions proposed in [1], e.g. the symmetric and asymmetric form with slope constraint conditions $P=1,2$ and $1/2$.

By taking into account the 'adjustment window conditions' $|i-j|\leq r$ two alternative parallel structures are proposed. Both parallel schemes can be implemented by using an array which contains $n$ processing elements, where $n$ is equal to the number of points on each column $i$, that is $n=2r+1$. The basic idea presented here, is that the partial sums related to the points of the column $i$ can be computed by the PEs in parallel, if the partial sums related to the points of the columns $i-1$ and $i+2$ are known.

II. DTW USING A CIRCULAR ARRAY

In order to describe the first scheme, let us refer to the example of

---------------------------------
National Technical University of Athens, Division of Computer Science, Zografou, 15773 Zografou, Athens, Greece.
The basic building blocks of each PE $p(k)$ are the following:

1. Memory unit consists of two memories $Ma, Mb$ whose sizes are equal to 1, where 1 is the size of the feature vectors $a_i, b_j$.
2. Registers $S0, S1, S2, S3, d1, d2, d3$ which retain the partial sums $S_t(k), S_t(k-1), S_t(k-2)$ and the local distances $d_t(k), d_t(k-1)$ respectively, where the index $t$ is referred to the computation step. This is the state of processors at the end of step $t$.
3. An arithmetic unit performing whatever operations are required.
4. A control unit and one output channel for I/O transfers (from $p(k-1)$ and to $p(k+1)$) and two ports connected to RPB and TPB buses.

In the beginning of step $t$, the processors have in their own memories $Ma, Mb$ the feature vectors $a_i, b_j$ (which have been memorized during the previous steps). Required for the computation of the local distances of column $i$. At step $t$, they perform the following operations in parallel.

- a) They compute the local distances $d_t(k)$ from the contents of registers $Ma$ and $Mb$ and the result is loaded into register $d1$, while the previous content of $d1$ is shifted to register $d2$.
- b) They read the new feature vectors $a_i$ (from RPB bus) and store it in their own memories $Ma$.
- c) The content of $d1$ of $p(k)$ is transferred to $d3$ of $p(k+1)$, $k=0,1,2,...,n \mod n$.
- d) They compute the partial sums $S_t(k)$ from the contents of registers $S1, S2, S3, d1, d2, d3$, which contain the partial sums $S_t(k-1), S_t(k-2)$ and the local distances $d_t(k), d_t(k-1)$ respectively, and store the result in $S0$. This computation is based on the following equation:

$$S0 = \min\{((S3)+2(d3)+d1), ((S1)+2(d1)), ((S2)+2(d2)+d1)\}$$

where $(S)$ indicates the value of register $S$.
- e) The content of $S0$ is transferred to $S1$ of $p(k+1)$, while the previous content of $S1$ is shifted to $S2$.
- f) The content of $S1$ of $p(k)$ is transferred to $S3$ of $p(k+1)$.

Hence, the registers of each PE retain the partial sums and those local distances required for step $t+1$, as mentioned above.

Some modifications on the operation of processor $p(k)$, which computes the partial sum related to the point $C(i,j)$, where $j=i-r$, must be made.

1. It has to absorb $b_{i-1}$ (from TPB bus), since at the next step it will compute the partial sum related to the point $C(i+1,j)$, $j=2r+i+1$. For instance, this input instruction can be pipelined with computation $d$ (since at this stage the PE computes the partial sum from (1.b) and (1.c), while the others from (1.a), (1.b) and (1.c)).
2. At stage e) it sets $S2$ to the highest value (since at step $t+1$ it will compute the partial sum related $S(i+1,j), j=i+2r+1$ from equations 1.a and 1.b). In order to implement the computation scheme cyclically, as shown in fig.2.a, the above PE will function similarly after $2r+1$ steps, when it will compute the partial sum related to the point $C(i+2r+1,j+r+1)$. For instance, this can be achieved by a control line which generates a signal. At each step this signal is rotated in the ring of fig. 2.b. It passes from $p(k)$ to $p(k+1), k=0,1,n \mod n$ and controls the function of the PEs.
III. DTW USING A LINEARLY CONNECTED ARRAY

For the second parallel solution, depicted in fig. 3.a (as in the previous example r has been set equal to 2), the partial sum $S(i,j)$ for point $C(i,j)$ is computed by processor $p(k)$, where the index $k$ satisfies the eq. $k = (j-i) + r + 1$. This parallel solution requires an array, as shown in fig.3.b, where each processor $p(k)$ can communicate with processors $p(k+1)$ and $p(k-1)$, $k = 1, 2, ..., 2r+1$. In this case, each processor consists of a memory unit, a control unit, an arithmetic unit, 3 input and 2 output channels for I/O transfers and a number of registers. Let us refer to the registers of each PE, in order to clarify the scheme of operations performed by the array. The registers $S_0, S_1, S_2, S_3, d_1, d_2, d_3, d_4$ of each PE retain the partial sum $S_k(k), S_{k+1}(k), S_{k+1}(k+1)$ and the local distances $d_4(k), d_4(k-1), d_4(k+1)$ respectively. This is the state of registers at the end of step $t$. In the beginning of step $t$, PEs $p(k)$ $k = 1, 2, ..., 2r+1$ have in their own memories $Ma, Mb$ the feature vectors $a_i, b_j$ (which have been memorized by $p(k)$ at the step $t-1$), required for the computation of the local distances of column $i$. At step $t$, the operations performed in parallel by the PEs, are briefly given in the following:

a) Each $p(k)$ computes the local distance $d(i,j)$ or $d(k)$ related to the point $C(i,j)$ of the column $i$, where $k = j-i+r+1$ and stores the result into $d_1$. b) Each PE reads the new feature vector $b_j$ (which will be used for the next step) and stores it in memory Ma. c) The contents of the buffer Mb of $p(k)$ are transferred to the buffer Mb of $p(k+1)$. Processor $p(2r+1)$ reads the new feature vector $b_j$ shown in fig. 3.b. d) The content of $d_1$ of $p(k)$ is transferred to $d_2$ of $p(k+1)$ and to $d_3$ of $p(k-1)$, while the previous content of $d_3$ is shifted to $d_4$. e) Each PE computes the partial sum $S_4(k)$ related to the point $C(i,j)$, from the contents of $S_0, S_1, S_2, d_1, d_2$ and $d_4$, which contain the partial sum $S_4(k), S_{k+1}(k), S_{k+1}(k+1)$ and the local distances $d_4(k), d_4(k-1)$ and $d_4(k+1)$. The result of the computation is loaded into $S_0$. f) The content of $S_0$ of $p(k)$ is transferred to $S_1$ of $p(k+1)$ and to $S_2$ of $p(k-1)$, while the previous content of $S_2$ is shifted to $S_3$.

The result of the DTW algorithm $S(i,j)$, can be found in $S_0$ of processor $p(k)$, where $k = (j-i)+r+1$.

IV. PERFORMANCE DISCUSSION

The number of steps required by the arrays to implement the algorithm is $I+r+1$ ($r+1$ steps are required for the initialization). Furthermore, it is clear that the operations performed by the arrays can use parallelism and pipeline. (For instance, some computations and transfers can be performed in a parallel fashion). Using these techniques, the speedup $Sp$ and the efficiency $Ef$ of the arrays is approximately $Sp = M / (I+r+1)$ and $Ef = M / (I+r+1)(2r+1)$, where $M$ is the number of points in the search area (fig. 1). The quantity $M$ is given by

$$M = (I-1)(I-r-1)(I-r-2)(J-r-1)(J-r)/2$$

(3)

The array proposed in [3] can achieve a speedup of 200, if one processor is positioned at each point in the search area (fig.1). For typical values of $I=40$ and $r=7$, the efficiency of such an array is 0.36. For the same values, the efficiency of the arrays proposed in this paper is 0.7, which is twice the efficiency of the array presented in [3]. Furthermore, the speedup is greater than that of single diagonal design [2] arrays (since the number of steps in such schemes is proportional to the number of diagonals $I+J$), while the efficiency is similar to the efficiency of those structures.
V. DTW USING A SMALL NUMBER OF PEs

Let us now consider a more general case, where the number \( n \) of processors is less than \( 2r+1 \). In such a case, two solutions related to the previous parallel schemes can be obtained. The interconnection network of these structures is similar to those of fig. 3.b and 2.b, but the memory requirement for each PE is proportional to \( m \) (\( m \) is the number of points, the partial sums of which are computed by each PE at column \( i \)). These schemes are illustrated in fig. 4.a and 4.b. The quantity \( r \) has been set equal to 3 and \( n=2 \) PEs are used. At every step \( t \), the PEs compute the partial sums related to the points \((i,j)\), as shown in fig. 4.a and 4.b.

VI. CONCLUSION

Some DTW algorithms can be implemented in parallel with any number of processors lower or equal to \( 2r+1 \). The resulting architectures are homogeneous. Vertical scanning of the constrained DTW area is preferred here to the diagonal scanning generally used. Very high performance is obtained by the above technique. A more detailed study of the parallel approach in DTW is given in [4].

REFERENCES