A systolic machine for string correction

Patrice Frison, Dominique Lavenier

INRIA

UNITÉ DE RECHERCHE
INRIA-RENNES

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cédex
France
Tél. (1) 39 63 55 11

Rapports de Recherche

N° 780

A SYSTOLIC MACHINE FOR STRING CORRECTION

Patrice FRISON
Dominique LAVENIER

DECEMBRE 1987

IRISA

INSTITUT DE RECHERCHE EN INFORMATIQUE
ET SYSTÈMES ALÉATOIRES

Campus Universitaire de Beaulieu
35042 - RENNES CÉDEX
FRANCE
Téléphone : 99 36 20 00
Télex : UNIRISA 950 473 F
Télécopie : 99 38 38 32

A SYSTOLIC MACHINE FOR STRING CORRECTION

UNE MACHINE SYSTOLIQUE POUR LA CORRECTION DE CHAINES

Patrice Frison, Dominique Lavenier

Décembre 87

Publication interne n° 384 - Décembre 87 - 18 pages

Abstract

Dynamic programming techniques using Levenshtein metric concept are particulary efficient for string correction. However, the method may not be implemented on conventional computer when large vocabularies or data base environment are required. Fortunately, dynamic programming methods are good candidates for parallel implementation and systolic architectures seem to be an interesting structure. A systolic string correction machine is presented in this paper. The proposed systolic machine has been designed as a peripheral device connected to an host computer and is able to manage efficiently string operations. The systolic array is implemented with 18 VLSI prototype chips connected as a linear regular structure. The first application under implementation deals with the real-time keyboard mistyping correction problem. First results show that a speed-up factor of about 60 may be obtained compared to an IBM-PC AT implementation.

Résumé

Les techniques de programmation dynamique sont particulièrement efficaces pour la correction de chaînes. Cependant, lorsque de gros dictionnaires ou d'importantes bases de données sont mis en jeu, l'implémentation sur des processeurs traditionnels devient inadaptée. Par contre, leur parallèlisation sur des architectures de type systolique semble être une voie particulièrement prometteuse. Une machine dédiée à la résolution de tels problèmes est présentée. Elle peut être considérée comme périphérique d'un processeur hôte spécialisé et performant pour le traitement de chaînes. Le réseau systolique intègre 18 processeurs VLSI interconnectés linéairement. La première application en cours de développement vise la correction interactive de fautes de frappe au clavier. Les premiers résultats indiquent un gain en vitesse de l'ordre de 60 par rapport à un IBM-PC AT.

1 INTRODUCTION

The increasing use of computer systems for the handling of textual data has led to a great deal of interest in software for the detection and correction of spelling errors [Hall80]. Many algorithms have been developped using different techniques to correct or retrieve erroneous informations.

The abbreviation method [Blai60], [Davi62], [Dame64], [Durh82], [Bick87], consists in finding the best reduced and significant sequence of characters to encode a word. Their techniques differ only in the way of forming the abbreviations and generally cannot be used for large vocabularies.

A method based on the analysis of the digram and trigram frequencies [Morr75], [Ange83] provides good results over important vocabularies but has the major drawback that all the errors may not be detected. Spelling correction using probabilistic methods has also been investigated by [Kash84].

The most successful approaches use the concept of the Levenshtein metric [Hall80]. The Levenshtein distance between two strings is defined as the minimum number of edit operations required to transform one string into another. It has been introduced by Wagner and Fisher in 1974 [Wagn74], [Lowr75] and has influenced many other researchers [Okud76], [Vero87]. The main advantage using this method is its generality, its flexibility and the good results it gives. Moreover, the Levenshtein distance may be computed efficiently by using a dynamic programming method [1].

In a computational point of view, it is very clear that the most sophisticated methods, are the most CPU time consuming. Furthermore, real-time error correction on a large vocabulary (10,000 words or more) by a general purpose computer is not feasible. In order to speed-up the execution of the algorithms, we propose to decompose the algorithm into elementary tasks to achieve multiprocessing and pipeling.

The dynamic programming methods are good candidates for parallel implementation on linear or two-dimensional array as demonstrated by [West83], [Yode82] and [Char86] for speech recognition methods [2]. Among the various parallel organizations that may be considered, systolic architectures seem to be very interesting structures. A systolic array [Kung82] is a special-purpose architecture made out of simple processing elements organized as a regular structure. Processors are locally connected, operate synchronously, and data circulate through the network

[1] In the rest of the paper we will focus essentially on this kind of method

[2] Note that phoneme based speech recognition algorithms are special types of string correction problems

in a pipeline fashion. The regular structure and operation of systolic arrays are particularly well suited for VLSI implementation, since the design of the whole network can be dramatically reduced to the design of only one processor.

The string correction machine that we propose is especially designed for investigating dynamic programming based correction algorithms. The main purpose of the machine is twofold. Firstly, we want to be able to evaluate different correction methods. For that purpose, we need to design a rather programmable machine, adaptable to a given problem. Secondly, we want a very fast machine that may give the results in real-time even with a large vocabulary. This point is very important, since in most cases a good string correction method evaluation is difficult to obtain due to the prohibitive CPU computing time of the experiments.

Our purpose in this paper is to present a prototype machine that we have already implemented. The machine is a linear systolic network connected to a host computer. It is under final validation tests and the first application programs are under development.

The next section presents the basic dynamic programming algorithm, its generalisation and how it can be parallelised in a systolic way. Section 3 describes the hardware implementation of the string correction machine. Section 4 describes how to use and program this machine. The last section gives some results about the main application under implemention : real-time keyboard mistyping correction.

2 STRING CORRECTION METHODS

In this section, we describe first the basic algorithm and the possible extensions we envision. Then we show how it can be computed on a two-dimensional systolic array. Finally, we show that the computation may be supported by a linear structure of processors.

2.1 Basic Algorithm

The main idea of the algorithm is to compare two strings of characters by computing a distance, called the edit distance, which represents the minimal cost to transform a string to another one by using elementary operations (the edit operations). The edit operations that are more often considered are the following :

- substitution : one character is changed into another single character.
- insertion : one character is inserted to the given string.

- deletion : one character is deleted from the given string.

By using sequences of such operations, any string may be transformed into any other string. It is then possible to take the smallest number of operations required to change one string into another as the measure of the difference between them. In addition, a cost may be associated with the different edit operations, depending on the application under consideration.

More formally :
let $X = (x_1, x_2, ..., x_i, ..., x_n)$
and $Y = (y_1, y_2, ..., y_j, ..., y_m)$ be two strings to compare ;
let $d(x, y)$ be the cost of an edit operation to change x into y ;
let $\wedge$ represent the nul character.

It has been shown [Wagn74] that the edit distance $D(n, m)$ is obtained from the following recurrence relation :

$$D(i,j) = Min \begin{cases} D(i-1, j-1) + d(x_i, y_j) \\ D(i-1, j) + d(x_i, \wedge) \\ D(i, j-1) + d(\wedge, y_j) \end{cases} \quad (1)$$

with the initial condition $D(0,0) = 0, D(i,0) = \infty$ and $D(0,j) = \infty$.

The edit operation $d(x_i, \wedge)$ represents the deletion case while $d(\wedge, y_j)$ represents the insertion case. The elementary edit operation $d(x, y)$ may have different values depending on the characters considered. This point will be examplified in section 4.

2.2 Generalization

The basic relation can be extended in many ways for adapting to specific applications. In the following, we give two examples :

- Consider the case of the transposition of two adjacent characters for keyboard mistyping correction, commonly called the reversal edit operation. Many authors modelize this kind of error by a sequence of a single insertion and omission. However, when the edit operations are weighted by some coefficients in order to take into account the occurrence frequency of each edit operation, this simple model means that a reversal edit operation is very badly penalized. Another way to proceed is to add to equation 1 the minimization term :

$$D(i-2, j-2) + \gamma_t(x_{i-1}x_i, y_{j-1}y_j) \quad (2)$$

where γ_t is a function defined as follows :

$$\gamma_t(x_{i-1}x_i, y_{j-1}y_j) = \begin{cases} k_t & if\ x_{i-1} = y_j\ and\ x_i = y_{j-1} \\ \infty & if\ x_{i-1} \neq y_j\ or\ x_i \neq y_{j-1} \end{cases} \quad (3)$$

- In the phoneme based speech recognition area it may be interesting to include other minimization terms to modelize either the compression or segmentation of phonemes. The first one leads to the following term :

$$D(i-2, j-1) + \gamma_{cp}(x_{i-1}x_i, y_j) \quad (4)$$

the second :

$$D(i-1, j-2) + \gamma_{sg}(x_i, y_{j-1}y_j) \quad (5)$$

$\gamma_{cp}(x_{i-1}x_i, y_j)$ represents the cost of compressing the two phonemes x_{i-1} and x_i into a unique phoneme y_j.

$\gamma_{sg}(x_i, y_{j-1}y_j)$ represents the cost of segmenting phoneme x_i into the two phonemes y_{j-1} and y_j.

In the same way, other possible minimization terms may be found to modelize, for example, repetition or reduction of characters into a given string. Depending on the problem under consideration equation 1 will be tuned with specific minimisation terms. The next section shows the systolisation of such algorithms. For safe of clarity, only the basic equation is considered.

2.3 Systolic Implementation

The basic idea to compute equation 1 is to associate one processing element to the calculation of each value $D(i, j)$. Consider an array of processors denoted $P_{i,j}$ connected as indicated by figure 1. $P_{i,j}$ has three input ports I_v, I_h and I_d and three output ports O_v, O_h and O_d. 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 $P_{i,j-1}$.

We suppose that $P_{i,j}$ is able to perform the computation expressed by equation 1. Figure 1 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 1. These data are represented by solid arrows. $D(i-1, j-1)$ is produced by $P_{i-1,j-1}$, $D(i-1, j)$

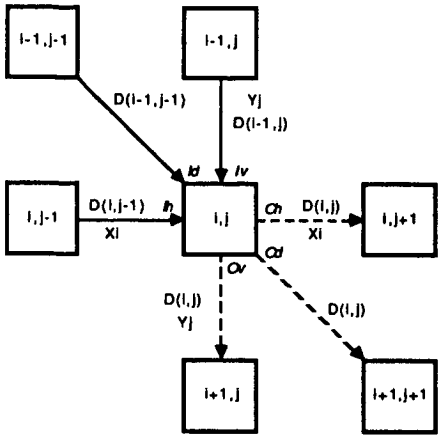

Figure 1: Inter Processes Data Flow

and y_j are produced by $P_{i-1,j}$ and $D(i,j-1)$ and x_i are produced by $P_{i,j-1}$. With all these informations $P_{i,j}$ calculates the edit distance $D(i,j)$.

$P_{i,j}$ has not only to perform the computation of equation 1, 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 figure 1. $P_{i,j}$ sends $D(i,j)$ to processor $P_{i+1,j+1}$, $D(i,j)$ and x_i to processor $P_{i,j+1}$ and finally, $D(i,j)$ and y_j to processor $P_{i+1,j}$.

The overall operation of the two-dimensional systolic array for this 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 starts at time 0, at time t, all the processors $P_{i,j}$ such that $i+j-1=t$ are active. In such a way, it can be easily checked that all the arguments needed for the computation of equation (1) have already been computed and have been routed correctly. The result $D(n,m)$ is provided by processor $P_{n,m}$.

Since only one diagonal of this network is active at a time, it is possible to reduce the two-dimensional array into a linear array. Each processor of this array has to perform the computations of either a row, a column or a diagonal of the two-dimensional array previously described. A row scheme requires n processors, a column scheme m processors and a diagonal scheme needs at most $n+m-1$ processors. The chosen projection may be directed by different criteria :

- minimum number of processors

- maximum calculation throughput
- minimum data flow through the network.

or a combination.

In order to illustrate the operation of a linear network, let us compare the operation of the two-dimensional array with the operation of an horizontal linear structure (row scheme) depicted on figure 2.

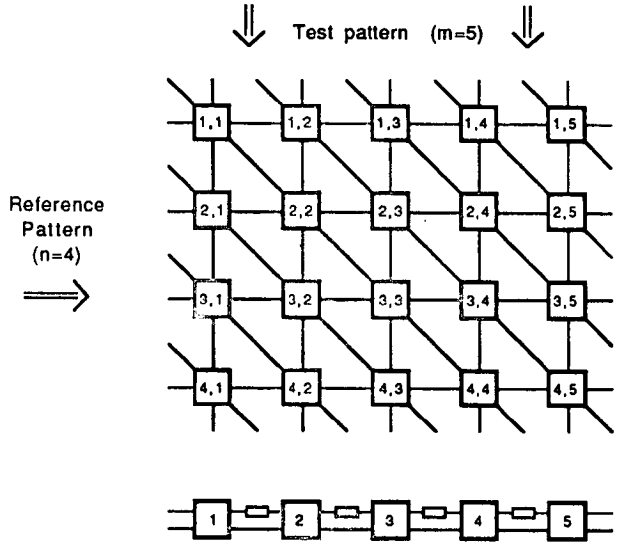

Figure 2: Two-Dimensional And Linear Array Structures

- On the two-dimensional array, let suppose that at $t = 1$ processor $P_{1,1}$ computes the distance $D(1,1)$ and sends it to its neighbors. At time $t = 2$, processors $P_{1,2}$ and $P_{2,1}$ compute respectively $D(1,2)$ and $D(2,1)$. $D(2,2)$ cannot be calculated during the same cycle because the informations required from $P_{1,2}$ and $P_{2,1}$ are not yet available. On cycle 3, $D(1,3)$, $D(2,2)$ and $D(3,1)$ can be computed. Note that $D(1,1)$ is used one cycle later by $P_{2,2}$ to produce $D(2,2)$.
- On the linear array, processor P_1 emulates the first column of the two-dimensional array, processor P_2 the second, processor P_3 the third, and so on. At $t = 1$, P_1 calculates $D(1,1)$ stores it and sends it to P_2. At $t = 2$, P_1 emulates $P_{2,1}$ and computes with the previously stored distance $D(1,1)$, the

edit distance $D(2,1)$. At the same time $D(1,2)$ is computed by P_2. The next cycle provides $D(3,1)$ on P_1, $D(2,2)$ on P_2 and $D(1,3)$ on P_3. Computing $D(2,2)$ means to have delayed $D(1,1)$ of one cycle. It is the purpose of the delayed registers inserted between the processors shown on figure 2.

3 HARDWARE IMPLEMENTATION

The string correction machine is a peripheral for high speed processing over large dictionaries. In the applications that we consider, a large memory is required to memorize dictionary words. Access to a word sequence must be dictated by the applications. A control processor has been designed to manage access to the dictionaries in order to feed data to the systolic processing machine. The machine is organized into two modules as shown on figure 3.

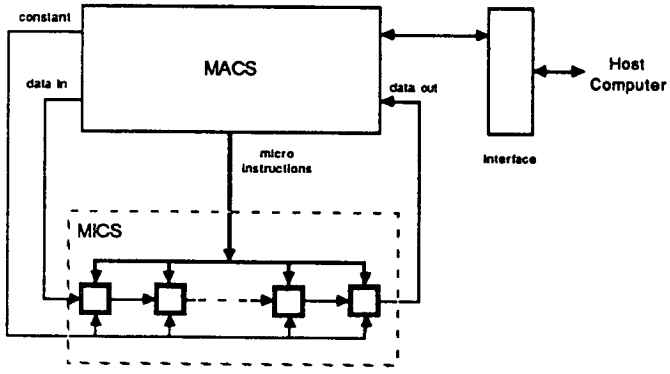

Figure 3: The String Correction Machine

The first one, called MICS supports the linear systolic processors. The second one, called MACS is in charge of generating data and microcode to MICS. In the next two sections, we present briefly these two modules.

3.1 The Systolic Module

3.1.1 Interconnexion Scheme

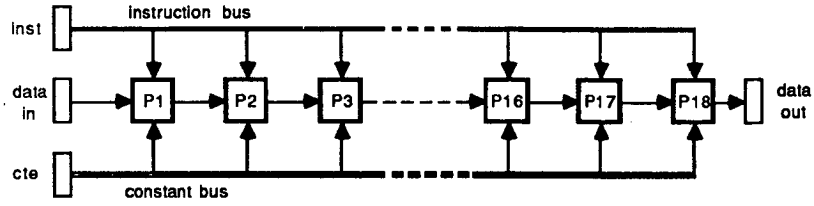

Figure 4: MICS Module

The MICS module is organized with 18 processor chips as described on figure 4. Each processor is connected to its two neighbors via two 16-bit ports. The third port of each processor (an input port) is connected to a single bus called the constant bus. This bus is used to broadcast the same value to the processors simultanously (in a non-systolic way). This facility simplifies programming of the machine in many cases.

Externally, the MICS module is accessed through four ports :

- an input port for delivering data to the first processor.
- an output port for obtaining data from the last processor.
- a constant port for providing all the processors with the same data.
- an instruction port to send micro-instructions to the processor array.

3.1.2 The Processor Element

The processor element used is a single chip called API89, a prototype chip that was designed for a speech recognition application [Fris85]. Externally, the API89 chip is provided with 3 16-bit ports : 2 input ports and 1 output port. This allows the designer to use the chip in either a two-dimensional or a linear systolic structure [Char86]. Internally, the processor contains 4 main modules connected to a single bus as shown on figure 5. These modules are :

- an arithmetic unit (AU) capable of performing elementary operations such as addition, substraction and incrementation on 16-bit values.
- an array of 16 general purpose registers.
- a 60 12-bit word memory.

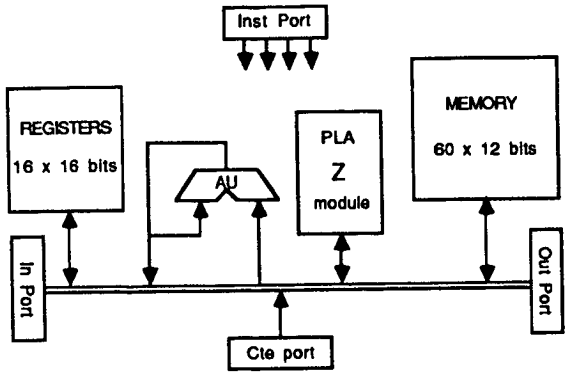

Figure 5: The API89 Processor Chip

- a lookup table called the Z module used for summing up values in a logarithmic representation [Fris85]. This last module is particularly interesting when probabistic calculations are processed.

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 non-overlapping clock scheme.

The operation of the circuit is controlled by 10-bit micro-instructions which are generated outside the chip and decoded inside using a very simple control unit. This kind of control scheme takes advantage of the systolic nature of the machine since the processors execute simultanously the same program and therefore can share a unique controller.

In order to increase speed, micro-instructions are latched in a pipeline register. In this way, instruction decoding and execution may overlap. Thus, one instruction is executed every clock phase.

3.2 The Control Module

The control module has three main functions :

- (1) generate instructions to the MICS module
- (2) communicate with a host computer

- (3) communicate with the MICS module

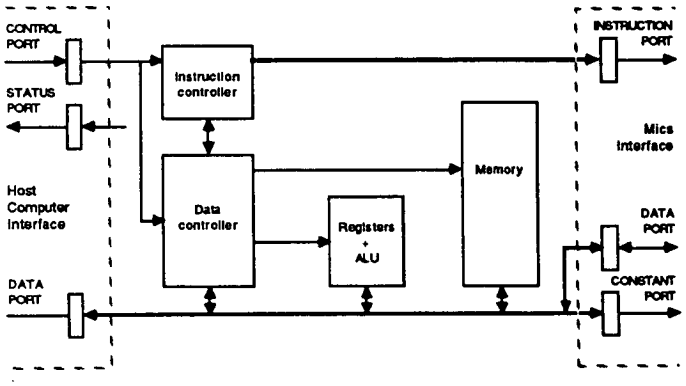

Figure 6: MACS Module

Two micro-program controllers are used for that purpose as shown on figure 6. The first one, called Instruction Controller is in charge of point (1). It is a very simple machine designed around a micro-program memory and a 2910 AMD micro-controller chip. The second one, called Data Controller takes care of points (2) and (3). It is a more complex machine. It contains a micro-program memory and a controller chip as well. It contains also a large memory to store dictionary data. This memory is accessed through an address generator to simplify regular addressing. Some registers and a comparator are also used in order to process data coming from the MICS module.

The host computer is able to load micro-programs to both controllers and therefore to configurate the machine for a dedicated application. When the dictionary memory has to be loaded, the Data Controller is used to receive and acknowledge the data coming from the host and to store them. This task may be time consuming, however, in many applications, the dictionary memory is filled only once.

4 PROGRAMMING THE MACHINE

In this section we will show how to use the machine to program a string correction algorithm.

4.1 Keyboard mistyping correction algorithm

To illustrate the operation of the machine on a real problem, we will focus on the mistyping correction problem. First of all, we need to give more details on the edit distance calculation. To be able to correct typing mistakes with good accuracy, the edit operation cost $d(x,y)$ is defined as follows :

$$d(x,y) = \begin{cases} 0 & \text{if } x = y \\ k_1 & \text{if } x \text{ and } y \text{ are two neighboring keys on the same keyboard line} \\ k_2 & \text{if } x \text{ and } y \text{ are two neighboring keys on two different keyboard lines} \\ k_3 & \text{otherwise} \end{cases}$$

$$d(x,\wedge) = k_4$$

$$d(\wedge,y) = k_5$$

Parameters k_i are constant values to weight the different edit operations. It is interesting to note that, in this way, the physical keyboard topology is taken into account for substitution errors. As a result, $d(x,y)$ may be represented by a square matrix M of the size of the alphabet (including the $\wedge$ character).

4.2 Parallel algorithm choice

In section 2.3 we have demonstrated that it is possible to compute the edit distance $D(n,m)$ between two words of size n and m respectively, in a parallel fashion using either n, m or $n+m-1$ processors depending on the way the calculation is done. In order to correct a mispelled word T, it is necessary to compute the edit distances between T and each word of the dictionary and memorize the smallest distances. The best approach is to use the linear array depicted on figure 2. It consists in assigning each letter T_i of the word T to the processor P_i and to feed the array with the words of the dictionary, one character per systolic cycle.

4.3 Initializing the systolic array

The first step to undertake is to load each processor P_j with the appropriate T_j character. Using the previous parallel algorithm, processor P_j has to calculate the values

$$D(k,j) \; for \; 0 < k \leq n$$

This calculation is done using equation (1). It is important to note that processor P_j does not need to know the whole M matrix since T_j is a constant

character. It only needs to memorize one column of M indexed by T_j. Therefore the second initialization step is to load each processor memory with the appropriate M column.

4.4 One word distance calculation

According to the flow-graph depicted on figure 1 and given that one processor is in charge of the calculation of one column of the graph, it is simple to see that a processor executes the following program :

Cell_Program

Init $D_v = 0$
Init $D_{prev} = 0$

REPEAT
Receive D_d
Receive D_h
Receive x

$$D := Min \begin{cases} D_d + M[x, y] \\ D_h + M[\wedge, y] \\ D_v + M[x, \wedge] \end{cases}$$

Send D_{prev}
Send D
Send x
$D_{prev} := D$
$D_v := D$
UNTIL $x = Marker$

$Result := D$

End_Program

In this program, the variables D_d, D_v, D_h are used to store the $D_{..}$ values received "diagonally", "vertically" and "horizontally" respectively as shown on figure 1. The variable D_{prev} is used to delay by one systolic cycle the just calculated D value. The $Marker$ value is used to check the end of a word and therefore to store the computation result. This program shows that the reference word characters (storedin x variable) flow through the processor, one per systolic cycle.

The interesting result is provided by processor m if m is the length of the test word T. In the next section we show how to compare T against the whole dictionary

4.5 Full dictionary processing

Once the *Marker* of the first word has reached a processor, the *Result* value just calculated is stored. In the application that we study, it is not necessary to keep all the results, but only the best ones. Therefore, when a result is obtained it is stored eventually in the best word list.

When the marker has been processed by processor P_1, it is clear that this processor is available for a new calculation. Therefore, the next word of the dictionary enters the network one letter per systolic cycle. In this way, the processors of the network are constantly active since the dictionary words are processed in a pipeline way. Note however, that processors P_i such that $i > m$ are useless since the test word T is only m characters long.

When the whole dictionary words have been processed, the results are delivered to the host computer.

5 SOME RESULTS

The algorithms presented were tested with real data in order to verify their performance. We focus on the mistyping correction problem using an alphanumerical keyboard. Several texts were typed very rapidely without possible corrections and computed to detect and correct the spelling errors. After calculations, a sequence of the 10 most probable correct words is proposed for words not found in the dictionary.

Equation 1 has been used including the transposition minimization term 2. The values of the different edit operations have been set as mentioned on section 4.1. The chosen parameters are listed below :

substitution
−correct key : 0
−left/right keys : $k_1 = 0.5$
−up/down keys : $k_2 = 0.8$
−other keys : $k_3 = 1$

deletion : $k_4 = 0.9$
insertion : $k_5 = 0.9$
transposition : $k_t = 0.6$

The first results are very promising since we get more than 95 % of success of finding the correct word in the first position with these arbitrary (manually) chosen parameters. Furthermore, in almost 100 %, the correct word belongs to the first 3 proposed words. Tuning these parameters more accurately [3] would certainly give better results. These simulations have been run over a vocabulary of 3,800 words.

Speed comparison has been made with the systolic string correction machine previously described and with other computers. The basic algorithm has been implemented using C language both on IBM-PC AT computer and a Sun 3/160 workstation. It takes about 10 seconds on a Sun machine to match a seven characters test pattern with a dictionary of 3,000 references and nearly 60 seconds on the IBM-PC. The systolic machine gives the result in less than one second.

Many improvements can be made in order to increase the algorithm efficiency. It is clever, for example, to sort the dictionary words as a function of the length of the words and then to scan only parts with approximately the same word length as the test input one. However, it is important to note that the efficiency of the dynamic programming techniques is such that very badly mistyped strings may be corrected anyway. Therefore, trying to reduce the scanning area in order to speed-up the algorithm may be a contradiction.

6 CONCLUSION

We have described a prototype systolic machine for fast string correction algorithms. It has been demonstrated that a parallel implementation of the commonly used dynamic programming techniques may be obtained on a linear array of pro-

[3] with the error frequencies associated to the operator for example

cessors. The systolic array is implemented with 18 VLSI chips communicating via 16-bit parallel ports. This array is controlled by a special module that generates micro-instructions to the array and also that handles the array inputs and ouputs.

With such a machine we showed that implementing a mistyping correction algorithm will give very high speed performances. These performances are mainly due to the fact that the machine combines both parallelism and pipeline techniques. Furthermore the fact that the machine is micro programmed contributes equally to its inherent speed. The speed-up obtained over conventional micro- (IBM-PC AT) or even mini- (SUN-3/160) computers is considerable since factors of about 60 and 10 respectively are obtained.

This machine is a testbed for string correction techniques and in that sense it is not a dedicated application device. With this machine we expect to explore different application domains with a fast response time. The applications involved include phoneme based speech recognition, regular expression search, DNA sub-string matching in molecular biology area.

References

[Ange83] R.C Angell, G. E. Freund, P. Willett, "Automatic spelling correction using a trigram similarity measure," *Infor. Proc. and Manag.*, vol. 19, n° 4, pp. 225-261, 1983.

[Bick87] M. A. Bickel, "Automatic correction to misspelled names : a fourth-generation language approch," *Comm. ACM,* vol. 30, n° 3, pp. 224-228, 1987.

[Blai60] C. R. Blair, "A program for correcting spelling errors," *Inform. Contr.*, pp. 60-67, 1960.

[Char86] F. Charot, P. Frison, P.Quinton, "Systolic Architectures for Connected Speech Recognition," *IEEE Trans on ASSP,* vol. 34, n° 4, pp. 765-779, 1986.

[Dame64] F. J. Damerau, "A technique for computer detection and correction of spelling errors," *Commun. ACM,* vol. 7, pp. 171-176, 1964.

[Davi62] L. Davidson, "Retrieval of misspelled names in an Airline's Passenger record system," *Comm. ACM,* vol. 5, n° 3, pp. 169-171, 1962.

[Durh82] I. Durham, D. A. Lamb, J. B. Saxe, "Spelling correction in user interfaces," *Comm. ACM* vol. 26, n° 10, pp. 764-773, 1983.

[Fris85] P. Frison, P. Quinton, "An Integrated Systolic Machine for Speech Recognition", in *VLSI : Algorithms and Architectures,* Edited by P. Bertolazzi and F. Luccio, North Holland, pp. 175-186, 1985.

[Glan57] H. T. Glantz, "On the recognition of informations with a digital computer," *J. ACM* vol. 4, pp. 178-188, 1957.

[Hall80] P. A. V. Hall, G. R. Dowling, "Approximate string matching," *Comput. Surv.*, vol. 12, pp. 381-402, 1980.

[Kash84] R. L. Kashyap, B. J. Oommen, "Spelling correction error using probabilistic methods," *Computer text recognition and error correction,* ed. by S. Srihari, IEEE Computer society press, pp. 272-279, 1984.

[Kung82] H. T. Kung, "Why systolic architectures," *Computer,* vol. 15, pp. 37-46, Jan. 1982.

[Lowr75] R. Lowrance, R. A. Wagner, "An extension of the string to string correction problem," *J. Assoc. Comput. Mach.*, vol. 22, n° 2, pp. 177-183, 1975.

[Morg70] H. L. Morgan, "Spelling correction in systems programs," *Comm. Assoc. Comput. Mach.*, vol. 13, n° 2, pp. 90-94, 1970.

[Morr75] R. Morris, L. L. Cherry, "Computer detection of typographical errors," *IEEE transaction on professional communications,* vol. PC-18, n° 1, pp. 54-64, 1975.

[Muth77] F. Muth, A. L. Tharp, "Correcting human error in alphanumerical terminal input," *Information Proc. and manage,* vol. 13, n° 6, pp. 329-337, 1977.

[Okud76] T. Okuda, E. Tanaka, T. Kasai, "A method for correction of garbled words based on the Levensthein metric," *IEEE Trans. Comput.*, vol. C-25, pp. 172-177, 1976.

[Oomm87] B. J. Oommen, "Recognition of noisy subsequences using constrained edit distances," *IEEE Trans. PAMI* vol. 9, n° 5, pp. 676-685, 1987.

[Pete80] J. L. Peterson, "Computer programs for detecting and correcting spelling errors," *Comm. assoc. comput. mach.*, vol. 23, pp.676-687, 1980.

[Vero87] J. Veronis "Phonographic versus typographic correction in natural language interfaces," *Fifth int. Symp. in Applied Informatics,* GrindelWald, Switzerland, Feb. 87.

[Wagn74] R. A. Wagner, M. J. Fisher, "The string to string correction problem," *J. ACM,* vol. 21, pp. 168-173, 1976.

[West83] N. Weste, D. J. Burr, B. D. Ackland, "Dynamic time warp pattern matching using an integrated multiprocessing array," *Computer,* vol. 15, pp. 37-46, 1982.

[Yode82] M. A. Yoder, L. J. Siegel, "Dynamic time warping algorithms for SIMD machines and VLSI processor arrays," in *Proc. 1982 Int. Conf. Acoust., Speech, Signal Processing,* Paris, France, May 1982, pp. 1274-1277.

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique