An integrated 2D systolic array for spelling correction

Dominique Lavenier

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

An Integrated 2D Systolic Array for Spelling Correction

Dominique Lavenier

N° 1987

May 1993

PROGRAMME 1

Architectures parallèles,
bases de données,
réseaux et systèmes distribués

ISSN 0249-6399

An Integrated 2D Systolic Array for Spelling Correction

Dominique Lavenier*

Programme 1 — Architectures parallèles, bases de données, réseaux
et systèmes distribués
Projet API

Rapport de recherche n ° 1987 — May 1993 — 15 pages

Abstract: This paper presents a fully integrated spelling co-processor for speeding up the character string comparison process. The chip we present is architectured around a banded 2-D systolic array consisting of 69 processors and is able to process more than 2 million of words per second. The high regularity of the chip has been exploited for investigating a design methodology based on the automated generation of a representative subcircuit : *the kernel*.

Key-words: spelling correction, dynamic programming, systolic array, VLSI implementation, design methodology

(Résumé : tsvp)

to appear in *Integration, the VLSI journal*

*lavenier@irisa.fr

Unité de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu, 35042 RENNES Cedex (France)
Téléphone : (33) 99 84 71 00 – Télécopie : (33) 99 38 38 32

Un réseau systolique 2D intégré pour la correction de fautes

Résumé : Ce rapport présente un co-processeur correcteur de fautes entièrement intégré permettant d'accélerer la comparaison de chaînes de caractéres. Le circuit est composé d'un réseau systolique 2-D tronqué de 69 processeurs et permet de traiter plus de 2 millions de mots par seconde. L'extrême régularité du circuit a été exploitée pour explorer une méthode de conception basée sur la génération automatique du circuit à partir d'une représentation de référence : *le noyau*.

Mots-clé : correction de fautes, programmation dynamique, réseau systolique, mise en œuvre VLSI, méthodologie de conception

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 [6]. As a matter of fact, texts commonly contain literals which represent up to 80 percent of the following errors:

- one letter is wrong,
- one letter is missing,
- one extra letter is inserted,
- two adjacent characters are transposed.

These errors are respectively referred as substitution, deletion, insertion and transposition errors. They are mainly due to keyboard mistyping during text edition. If detecting an erroneous word is quite an easy task, proposing suitable corrections generally requires complex processing.

The correction process consists in comparing the erroneous word with the words of a dictionary and keeps back only the words which have the best *likeness*. The difficulty is to compute this *likeness* in order to propose good corrections.

The method we use to determine likeness is based on dynamic programming techniques and provides extremely good results compared with other techniques, especially when erroneous words which have more than one spelling error are processed for correction [7]. Key location is one of the most important parameter involved in the correction process. For example, the erroneous word *progessor* will be corrected by *professor* instead of *processor* because the **f** key is nearer to the **g** key than the **c** key on a keyboard.

The major drawback of the algorithm comes from the amount of calculations required. With dictionaries of a few hundred thousands of words, the computation time on conventional personal computers, and even on recent workstations, can be very long and inacceptable for real-time spelling correction.

Parallel implementations based on dynamic programming algorithms on systolic arrays are very efficient. Many systolic architectures have been proposed to solve similar string correction problems such as speech recognition [1] [2] [3] or nucleic acid sequence comparison [8] [5]. But a solution integrating some tens of quite complex processors on a single VLSI chip was not feasible until the last few years. Today, the advancements of technology have made it possible.

The chip we propose includes a 69 processor array dedicated to spelling correction applications. It can be seen as a hardware accelerator (co-processor) for comparing one string (erroneous word) with a large set of references belonging to a dictionary. The result of the comparison process is a sequence of words which seem probable candidates for the correct word.

Systolic architectures are very regular structures and are well suited for VLSI implementation. The regularity can be exploited to help the designer during the different design steps, especially when the number of transistors becomes important and conventional CAD tools become inefficient.

The design methodology we used for designing the chip is based on the study of a *kernel* which has exactly the same structure as the desired circuit but where redundant structures are eliminated. In our case, it means, for instance, that for validating the implementation of the dynamic programming algorithm, one does not need to simulate it on the complete 69 processor array, but may use a smaller set.

The paper is organized as follows: sections 2 and 3 present respectively the method used for spelling correction (string comparison) and the implementation on a 2-D systolic array. Section 4 is dedicated to the chip architecture and section 5 describes the VLSI implementation. Section 6 is devoted to the design methodology. We conclude withe a performance evaluation of the chip.

2 String comparison algorithm

The rule we use for comparing two strings of characters is a measure of distance between these two strings. This distance, called the *edit distance*, is the minimal cost in terms of the number of operations required to transform one string into the other one using elementary operations such as substitution, insertion, omission or transposition. These are called *edit operations*.

By using sequences of such edit operations, any string may be transformed into any other string. It is then possible to take the smallest number of elementary edit operations required to change one string into another one as the measure of the difference between them.

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 be compared; let $d(x, y)$ be the cost of an edit operation to change x into y; let $\wedge$ represent the null character. It has been shown [9] [7] that the edit distance $D(n, m)$ is obtained from the following recurrence relation:

$$D(i,j) = Min \begin{cases} D(i-1,j-1) + d(x_i, y_j) \\ D(i-1,j) + d(\wedge, y_j) \\ D(i,j-1) + d(x_i, \wedge) \\ D(i-2,j-2) + \gamma_t(x_{i-1}x_i, y_{j-1}y_j) \end{cases} \quad (1)$$

where γ_t is a function obeying:

$$\gamma_t(x_{i-1}x_i, y_{j-1}y_j) = \begin{cases} K_t & if\ x_{i-1} = y_j\ and\ x_i = y_{j-1} \\ \infty & if\ x_{i-1} \neq y_j\ or\ x_i \neq y_{j-1} \end{cases} \quad (2)$$

with K_t representing the transposition cost. Initial conditions are:

$$D(0,0) = 0, \quad D(i,0) = D(i-1,0) + d(x_i, \wedge), \quad D(0,j) = D(0,j-1) + d(\wedge, y_j)$$

The edit operations $d(x_i, \wedge)$ and $d(\wedge, y_j)$ represent deletion and insertion respectively. The elementary edit operation $d(x, y)$ may have different values depending on the characters considered.

Figure 1 is a graphical representation of the edit distance computation between the test string *sytollic* and the reference string *systolic*. For the sake of simplicity, the permutation error is not represented. The insertion cost, the deletion cost and the substitution cost are equal to one. The total edit distance is then equal to 2 since there is one deletion and one insertion. The dashed line represents the optimal way to reach the solution.

The computation of equation (1) requires 3 minimizations and 6 additions, i.e. 9 elementary operations. Comparing two words of n characters requires $9n^2$ operations. With a dictionary of p words, the total amount of calculations for a correction process is equal to $9n^2p$ operations.

The complete French dictionary represents about 700 000 words when plural and conjugation forms are considered. If a word of 10 characters has been detected as erroneous it will be necessary to compare this string with all the words in the dictionary whose lengths are nearly similar. Taking words ranging from 8 to 12 characters implies comparing the string with approximately 300 000 target strings. The total number of operations is then equal to 270 million.

Real time spelling correction using dynamic programming techniques is thus computationally too expensive for conventional machines, especially when large dictionaries are involved. The dedicated structure we present in the next section offers an alternative solution.

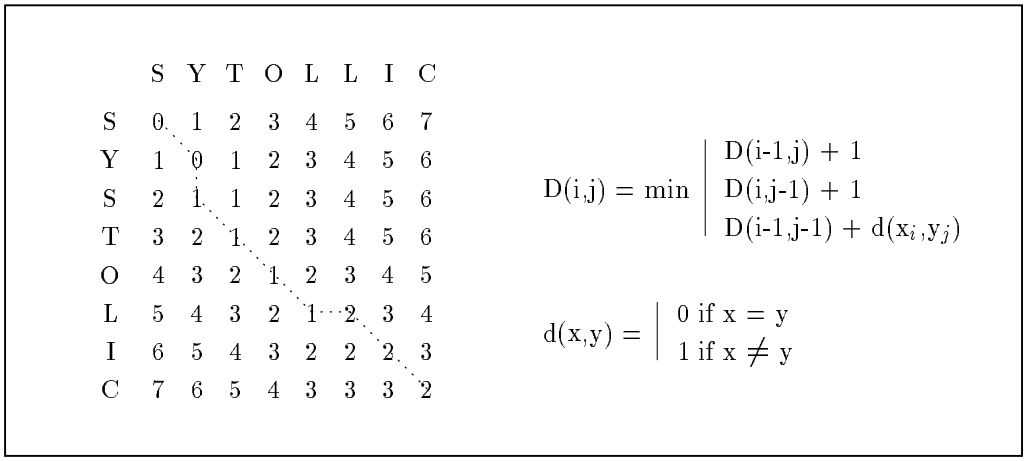

Figure 1: edit distance computation example

3 Parallel implementation on a systolic array

This section briefly describes how the equation of the previous section can be implemented on a systolic array. For the sake of clarity, we will only consider the model without permutation errors. The general architecture is not affected by this assumption while the explanations are greatly simplified.

3.1 Immediate implementation

Systolic implementation is based on assigning one processing element to the calculation of each value $D(i,j)$. Consider an array of n^2 processors connected as shown in figure 2. Processor $P(i,j)$ can read data produced by its neighboring processors $P(i-1,j-1)$, $P(i-1,j)$ and $P(i,j-1)$, and propagate its results to processors $P(i+1,j+1)$, $P(i,j+1)$ and $P(i+1,j)$. A systolic cycle performs an elementary distance calculation, that is to say : data acquisition, minimization according to equation (1) - without transposition errors - and data propagation.

On such a network, parallel execution of the comparison of two strings requires $2n-1$ systolic cycles, since an elementary distance computation $D(i,j)$ involves all processors $P(i,j)$ and processors run concurrently. This architecture is not optimal as the n^2 processors obtain only a speed of up to $2n-1$.

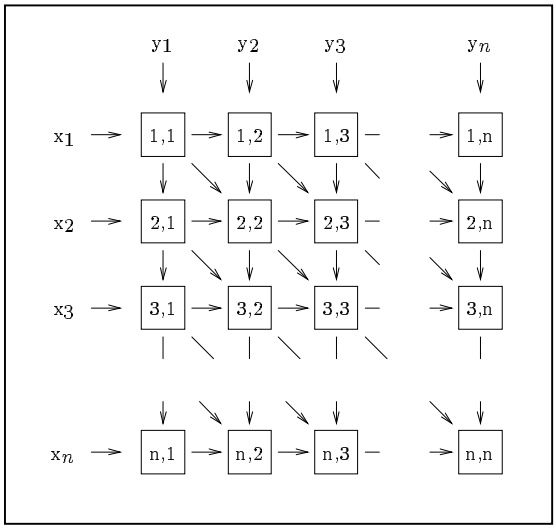

Figure 2: systolic network

3.2 Pipelined implementation

As a complete comparison lasts $2n-1$ systolic cycles, and as a processor is used during only one cycle, it is available during the other cycles for other tasks. Furthermore, the edit distance calculation at t_k is computed on a diagonal of processors consisting of processors $P(i,j)$ such that $k = i + j - 1$. Thus it is possible to start a new comparison on each systolic cycle.

Pipelining the string comparison in this way allows one to increase considerably the efficiency of the network, since once the array has been initialized, one comparison result is produced every systolic cycle. The speed up in comparison with a sequential machine is given by :

$$\Gamma = \frac{pn^2}{p+2n-2} \approx n^2 \qquad with \;\; p \gg n$$

This last approximation is valid in the applications envisioned since the number of references can reach hundreds of thousands items for strings of maximally 20 characters.

4 Chip architecture

4.1 Systolic array dimensions

Spelling correction implies processing character strings which are not very long. The average length of words found in a French or English dictionary is about 8 characters. This means that the number of words longer than 15 characters is quite low and does not require important computation resources. For these exceptional words, the correction process can be done on conventional machines.

A 2D systolic network of 15×15 processors seems to be a good compromise for this specific application. To be able to process strings of different lengths on a fixed 15x15 processor array, strings shorter than 15 characters are appended with special characters. In that way, results are always found at the same processor (see figure 3). Unfortunately, the resulting 225 processor array is too large to be integrated on a single chip. However, as explained below, the number of processors can be drastically reduced.

We observe that erroneous words are always corrected by words whose lengths are identical up to at most $\pm$ 2 characters. This means that all useful computations are performed on a diagonal of the array, and that processors which are located on the lower left and upper right corners do not participate actively in the final result. Tests have been made to compute the edit distance according to this observation. They do not show appreciable changes in the result and, in most cases, when differences were detected, the proposed corrections were better!

The systolic array we have implemented is shown on figure 3. It is a 2D banded array with five diagonals of processors. The main diagonal counts 15 processors; this gives a total number of processors equal to 69 ($15 + 2 \times 14 + 2 \times 13$).

4.2 Dataflow and memory management

Each erroneous string must be compared with a large set of reference strings. The data associated with the references are flowing horizontally through the array while data attached to the erroneous string are considered as constant values during the whole correction process. Consequently, to perform its local computation, a processor $P(i, j)$ needs to receive, on each systolic cycle, three intermediate distances from its neighbors and a character x_{ki} corresponding to the i^{th} character of the k^{th} reference.

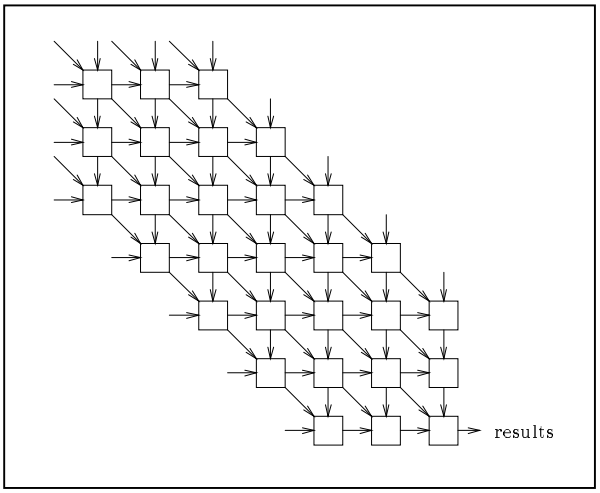

Figure 3: 2D truncated systolic array

Theoretically, costs associated with insertion and deletion errors depend on the nature of the characters. In practice, they are identical and can be replaced by constant values. So, the insertion and deletion costs $d(\wedge, y_j)$ and $d(x_i, \wedge)$, respectively, are replaced by two constants: Ki and Ko, respectively.

The reference dataflow implies that each processor must have a local table for storing substitution costs in order to be able to compute the first term of equation (1). As the characters of the erroneous string do not move during a complete dictionary comparison, the table stored into each processor can be reduced to be only the cost associated with a single character. The memory size is then equal to the number of characters in the alphabet; on the other hand, the content of the table inside the processors must be changed each time a new erroneous string must be corrected. However, downloading tables with new values takes a very short time compared to the whole correcting process.

Nevertheless, this solution consumes excessive space due to the fact that each processor contains its own table. A less expensive approach relies on two observations. Firstly the tables of all processors in one column are identical. Secondly, assuming that a systolic cycle takes several clock cycles (at least 5), a single memory element can be shared among processors since only one access is needed per processor, per systolic cycle. Therefore, the systolic array can be divided into 2 subarrays operating in parallel as shown in figure 4. The reference data array is a set of

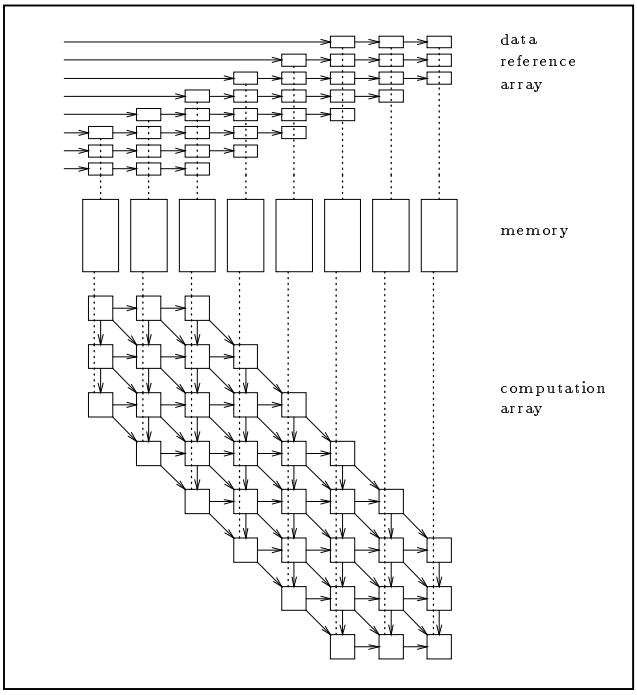

Figure 4: splitting the systolic network into 2 subarrays

registers containing the reference characters and which emulate the reference flow through the systolic network. The registers of a column are used sequentially to address the corresponding memory block and to obtain the appropriate substitution cost. The computation array is used to compute equation (1) in each cell, assuming that the distance cost is available in a register updated by the reference data array. This works correctly if the reference data array provides the reference data one cycle in advance of the computation (one systolic cycle ahead).

4.3 Memory implementation

As mentioned earlier, the memory size is equal to the number of characters of the alphabet and its content represents the substitution cost of two characters. This cost is determined by the proximity of the keyboard keys: the closer the keys are, the smaller the substitution cost is. By examining the cost table associated with a particular key, it can be seen that only a few values are significant (low cost)

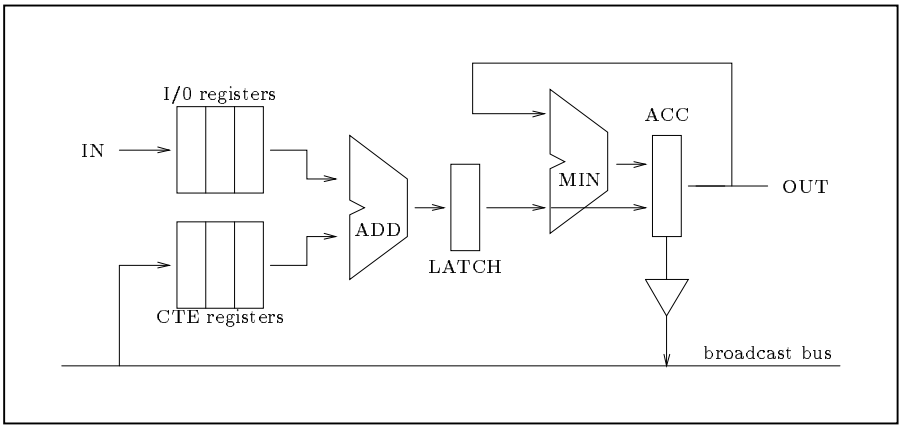

Figure 5: internal processing element architecture

and that they are associated with neighboring keys. Most of the other substitutions have a high cost and are quite similar. So, instead of systematically memorizing all costs, only interesting ones are stored using an associative memory. For other keys, a default value is substituted. In this way, memory size is considerably reduced and is independent of the alphabet size.

4.4 Processing element

The basic calculation performed by the processors of the computation array is the minimization of terms. Equations to be computed are of the form:

$$D(i,j) = Min \begin{cases} D(i-1,j-1) + Ks \\ D(i-1,j) + Ko \\ D(i,j-1) + Ki \end{cases} \quad (3)$$

where $D(i-1,j-1)$, $D(i-1,j)$ and $D(i,j-1)$ are results produced by the neighboring processors and Ki, Ko and Ks stand for insertion cost, omission cost and substitution cost, respectively. Ki and Ko are constant values while Ks depends on the reference string character.

Figure 5 sketches the processing element architecture. The **IN** input stores data (intermediate distances $D(i-1,j-1)$, $D(i-1,j)$ and $D(i,j-1)$) coming from the

other processors into a dedicated I/O register file data. The **CTE** input is connected to a broadcast bus for receiving constant data from the outside world and from the memory. These data are stored in a specific constant register file before being used.

As the arithmetic operations involved are very limited (addition and minimization), two specific units are implemented, namely an adder and a minimizer. These two units are pipelined due to the regular and repetitive structure of the computation. The accumulator can be loaded either from the adder or from the minimizer.

4.5 Chip control

The chip is programmable such that different applications based on equation (1) can be performed. The processors of the array are activated by micro-commands which control actions such as accumulator loading, I/O and CTE register selection, data acquisition, etc. These micro-commands are specified by an instruction which is received from the outside and decoded. In the same way, the memory actions (as well as data reference actions) are programmed depending on the calculation being performed.

Processor array, reference memory and data reference array thus operate synchronously since, they each execute one instruction every machine cycle. One instruction specifies actions to be realized concurrently on all these units. In that sense, the processor array can be considered to have an SIMD execution mode: all the cells are executing the same instruction at the same time.

5 VLSI implementation

The chip implementation has been realized having in mind the design methodology we wanted to investigate. Hence, we focus on designing a regular layout made only of a full custom cell array.

Figure 6 gives the floorplan of the chip. The systolic array is embedded in an orthogonal matrix such that diagonals are laid down horizontally. Furthermore, to get a very regular structure, dummy processors are added at the upper left and lower right corners. They are only used during initialization steps and do not participate to the edit distance computation. Similarly, the data reference array is also transformed into a regular rectangular matrix.

Each column of the processor array is connected to an independent associative memory by a bus, called the *broadcast* bus. The processors can input data from this

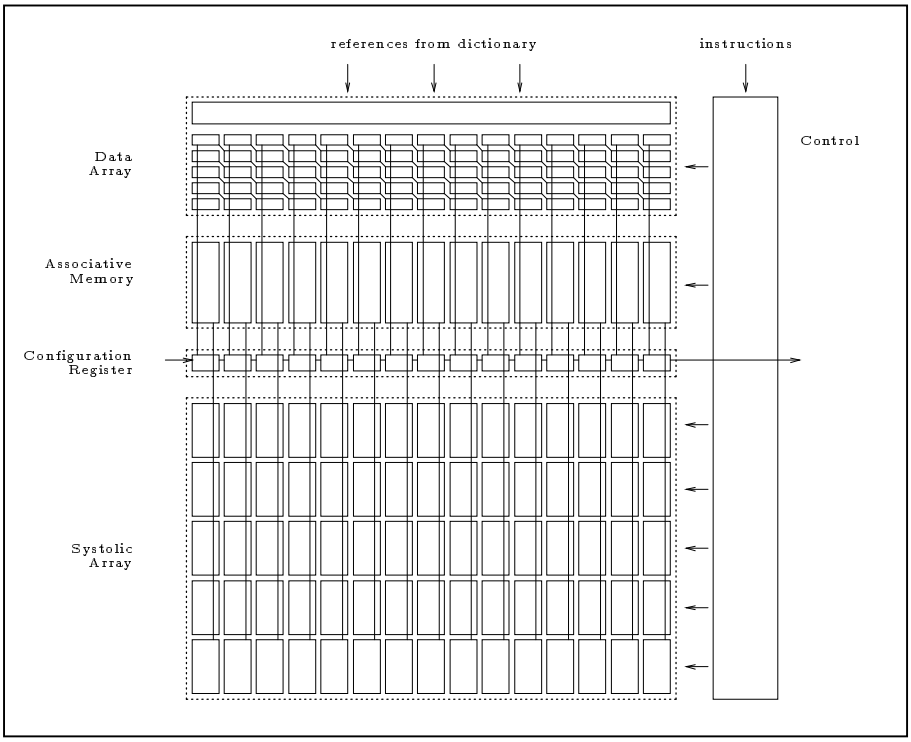

Figure 6: chip organization

bus and store them in their constant registers. Patterns are provided to the memory by the data reference array via another bus.

Before starting a whole comparison process, some constant registers of the processors, as well as the memory, must be initialized,. A configuration shift register has been added to perform these different tasks. Data are fetched from the outside world on the left-hand side register and can be read on the right-hand side. As an example, the memory is loaded by first reading 15 different data on the input configuration shift register port, and then pushing these data onto the memory input port. This operation is repeated until the memory is full.

The configuration register also provides a nice way to test the chip functionality: as all the different units are connected to this mechanism, data can be sent to and received from any part of the chip core. In order to be able to pick up data from every processor, and then to test each processor locally, the micro-command which allows a processor to write its content on the broadcast bus specifies the processor

row. Consequently, SIMD execution mode of the processor array is not respected when performing this action.

Each module has been designed as an array of full custom cells, placed side by side. We obtain a compact structure without any global routing connections. The final circuit is a direct assembly of the five modules (processor array, data reference array, associative memory, shift register and decoder). The number of transistors of the chip is approximatively equal to 300,000.

6 Realization

Due to the large number of transistors in the design, conventional CAD tools become inefficient, especially when validation tasks have to be performed on full custom chip. For instance, the layout/schematic comparison step is one of the most important in the design of a full custom chip, since it is the only way to make sure that the layout represents correctly the schematic. Commercial CAD tools cannot generally carry out this validation step when the number of transistors is large.

The design methodology we developed is based on the study of a subcircuit, called the *kernel*. The idea is that the chip can be parametrized and built automatically. This is done by a generator which, according to different parameters, can produce either the real circuit or the kernel. The assumption is that if the kernel is correct then the real circuit is correct too (we suppose also that the generator contains no bugs !). The advantage is that the kernel may represents only a few thousands of transistors and, consequently, it can be designed and easily checked using standard CAD tools.

The properties of the kernel must respect the properties of the final circuit. This includes both topological properties and functional properties.

The topological properties concern the juxtaposition of the cells. For instance, the design rule checking of a N $\times$ N matrix of identical cells does not need a verification on the full area, but only on a 2 $\times$ 2 matrix. For more complex structures with different cells one must ensure that all the cell juxtapositions are represented inside the kernel.

From a functional point of view, rules which allow one to determine a kernel from the real circuit are not as precise and depend essentially on the kind of verification which is envisioned. The subset of features we have chosen allows the implementation of the basic dynamic programming algorithm and, then, allows the logical validation of the different hardware mechanisms developed.

RTL simulation has been done on the kernel schematic only. The kernel represents about 18 000 transistors, a number of components which is easily managed with conventional simulators. This reduced schematic (as compared with the schematic of the real circuit) permitted saving time on schematic capture and simulation.

If simulation time depends greatly on the transistor count of a chip, time spent for layout verification is one (or more) orders of magnitude higher. The most time consuming step is the comparison of the schematic versus the layout. In the case of circuits including a few hundreds of thousands of transistors, using such a method of verification is no longer realistic. The advantage of our approach is that the verification could be done with a reasonable number of transistors and could be performed with conventional full-custom CAD tools.

In order to validate our design methodology, a chip has been realized. Actually, the chip we realized is not the 69 processor array described in this paper, but a reduced version of 34 processors (a 12×12 matrix of processors with 3 diagonals). It was designed in a 1.5 μm CMOS technology. The chip contains 150 000 transistors and requires 50 mm^2 of silicon. The chip functions correctly, and has a maximum clock frequency of 12 MHz. Tests have shown that the clock frequency is presently limited by the input clock pad which has to drive a too large capacitance. The clock frequency could be easily increased up to 25 MHz (the response time of the adders and the minimizers are less than 40ns) by connecting, for example, two or more input pads in parallel to the clock wire.

A correction process without transposition errors involves a systolic cycle of 8 machine cycles (640ns). Since an edit distance result is available every systolic cycle, more than 1.5 million strings ($\leq$ 12 char) can be processed per second. In other words, when all processors are operating, a 400 Mops peak performance is achieved. If we now make the assumption that we have the 69 processor array and a clock frequency of 25 MHz, the dynamic programming algorithm (with transposition errors) needs a systolic cycle of 12 machine cycle (480ns). That means the chip could proccess more than 2 million strings ($\leq$ 15 char) per second and could reach a 1.3 Gops peak performance.

Full custom cells have been developed using the CADENCE environment and automatically assembled with the *Make-Array* tool. This tool needs a text file which represents the cell array description. The layout generator we designed produces such a text file description. It is a C program which takes parameters from a text file (edited manually by the designer), and generates another text file (the cell array description).

The layout generator takes care of electrical features of the chip; it automatically sizes the power amplifier transistors which control the micro-commands, according to the size of the processor arrays.

7 Conclusion

Generating a chip is a very fast process once the *kernel* has been validated. First one runs the cell placement generator with the desired parameters, then the *Make-Array* tool. It takes about 5 minutes to get the chip core and a lot more minutes to connect the chip core to the I/O pads.

Our investigation into the design of chips which have very regular structure is mainly based on the automated generation of a subcircuit, the kernel, for decreasing the time for conception. The first attempt was successful but there are improvements which can be made, especially for generating the kernel. In the present chip, the properties of the kernel were set manually by the designer and were assumed to be correct. The layout generator which is a simple C program is also assumed to be correct.

Future work will concern better tools for generating a representative kernel. The kernel used for checking the design rules could have been different than the kernel actually used. Instead of giving parameters to get a subcircuit (which we assume to have good topological properties), it should be better to have tool which could derive automatically the kernel from the actual layout. The use of 2D grammars to derive or to prove the kernel. may be an interesting field of research

Work concerning functional validation of properties of the kernel is currently under investigation. It uses a high level programming language, ALPHA [10]. An ALPHA program specifies algorithms with mathematical equations and provides mechanisms for deriving regular architectures. The transformations which can be done from the initial specifications to the final architecture are independant of parameters such as the size of the array or the number of diagonals. One may first specify the desired circuit and then uses ALPHA to apply a serie of transformations to obtain the target architecture. Then using the same transformations on another set of parameters, the kernel representation may be generated and used to do verification.

References

[1] J.P. Banatre, P. Frison, P. Quinton, "A network for the detection of words in continous speech," *VLSI81 int. conf.*, J.P. Gray Ed., Academic Press, 1981.

[2] D. J. Burr, B. D. Ackland, N. Weste, "Array Configurations for Dynamic Time Warping," *IEEE Trans. on acoustic, speech and signal processing,* vol. ASSP-32 n° 1, pp. 119-127, feb. 1984.

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

[4] 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.

[5] M. Gokhale & al., "SPLASH : A Reconfigurable Linear Logic Array", *Technical Report SRC-TR-90-012*, Supercomputing Research Center, Maryland, April 1990.

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

[7] D. Lavenier, " MicMacs : un réseau systolique linéaire programmable pour le traitement de chaînes de caractères," *Thèse de l'université de Rennes 1,* Juin 1989.

[8] D.P. Lopresti, "P-NAC : a systolic array for comparing nucleic acid sequences," *IEEE Computer,* vol 20, pp. 98-99, July 87.

[9] 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.

[10] C. Dezan, H. Le Verge, P. Quinton, Y. Saouter, "The ALPHA DU CENTAUR Environment," *International Workshop Algorithms and Parallel VLSI Architectures II*, ELSEVIER, pp. 325-334, june 1991.

Unité de recherche INRIA Lorraine, Technôpole de Nancy-Brabois, Campus scientifique,
615 rue de Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unité de recherche INRIA Rennes, IRISA, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unité de recherche INRIA Rhône-Alpes, 46 avenue Félix Viallet, 38031 GRENOBLE Cedex 1
Unité de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unité de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
ISSN 0249-6399