IAPR Workshop on CV - Special Hardware and Industrial Applications OCT.12-14, 1988, Tokyo

DESIGN OF TWO SPECIALIZED "REAL TIME" TEXTURE ANALYZERS AND COMPARISON : SYSTOLIC VERSUS NON-SYSTOLIC ARCHITECTURE

Gérald Ouvradou, Dominique Barba***

(*) ENSTB, BP 832, 29285 BREST (FRANCE)
(**) IRESTE, La Chantrerie CP 3003, 44087 NANTES Cedex 03 (FRANCE)

ABSTRACT

This paper deals with the design of a processor dedicated to a new and original method of texture feature extraction called *the Method of Curvilinear Integration*. The latter has exhibited some interesting features in the field of image segmentation (efficiency, computation saving [RONS]). In the first part, the method is analyzed from the point of view of complexity reduction and concurrent processes implementation. Then a convenient pipelined architecture is derived. The design of the latter is examined in terms of cost and performance. The next section deals with a systolic implementation of the method which offers great benefits for dedicated design but requires, in our case, some concessions on computation saving to be made. The paper ends with a comparison between these two approaches.

1 INTRODUCTION

This paper deals with the design of a processor dedicated to a new and original method of texture feature extraction called *the Method of Curvilinear Integration (M.C.I.)*. With regard to most texture analysis methods, the M.C.I., due to its computation saving, offers an opportunity to aim a low-cost real-time texture analyzer, providing a suitable dedicated processor is designed.

In the first section of the paper, the method is presented from a formal point of view. Then, in the second section, complexity aspects are taken into account in order to establish a computation saving sequential algorithm. The next section is devoted to a highly parallel hardware implementation of this cost effective algorithm. A pipeline approach is well adapted to the latter due to its low-level nature (regularity, repetitivity, fine grain underlying parallelism). Nevertheless, the complexity reduction reinforced the data dependence of the associated execution scheme. This feature introduces severe constraints from the parallelism point of view and requires some extra cost in hardware implementation (dynamically configurable architecture). This is why the last section deals with the implementation of a less critical but more computation demanding algorithm release. The paper ends with a comparison of the two proposed designs from the point of view of cost and performance taking into account utilization criteria.

2 FORMAL DESCRIPTION OF THE METHOD OF CURVILINEAR INTEGRATION

Let $Z(i,j)$ be the luminance function of a digitized picture. The M.C.I. associates an 8 dimension vector $T(i,j)$ to a pixel of the picture, each component of this vector being related to a scanning direction originated from the pixel. The set of scanning directions is uniformly distributed in $[0, 2\Pi[$. The method proceeds by curvilinear integration of the luminance function and returns a $T(i,j)$ component ; the value of this component is the distance from the pixel at which the computed value reaches a predefined threshold μ. In the mean time, a *watch-dog* is provided in order to stop the computation process if the threshold is not triggered after a predefined number of iterations. This secondary parameter is named kmx.

Let us now formalize the method with respect to a polar coordinates system $[r, \theta_n]$. In order to unify the formalization of computations related to orthogonal ($\theta_n = n\,\Pi/4$, $n = 0, 2, 4, 6$) and diagonal ($\theta_n = n\,\Pi/4$, $n = 1, 3, 5, 7$) orientations, this coordinates system is mapped onto the sampling square grid of the image. The relationship between a $n\,\Pi/4$ oriented displacement $(\Delta i, \Delta j)$ in the discrete image plane and the related variation Δr in our coordinates system is $\Delta r = \mathrm{MAX}(|\Delta i|, |\Delta j|)$. In this way, r takes only positive integer values during the scanning process. The related euclidian distance $\Delta\rho$ is restored by the expression $\Delta\rho = \Delta r\, P(n)$ with the function $P(n)$ defined on integer values by $P(n) = 1$ for n even and $P(n) = 2^{1/2}$ else.

Let us now consider a scanning direction θ_n ($n = 0, 1, \ldots, 7$) issued from a pixel O. Let $Z_n(r)$ be the luminance function referenced to the (O, r, θ_n) coordinates system (see figure 2.1). A scaling factor λ between spatial displacement and the luminance is introduced in order to compute curvilinear integration of the latter. Consider now a unitary displacement from $(r-1)$ to r, the associated value $\Delta S_n(r)$ involved in the curvilinear integration step is given by

$$\Delta S_n(r) = (\lambda^2 P(n)^2 + (Z_n(r) - Z_n(r-1))^2)^{1/2} \qquad (2.1)$$

The curvilinear integration has to be expressed as a two index function $S_n(r, k)$ with r the modulus of the starting pixel and k the iteration number. Thus one computes the curvilinear integration by the following recurrent formula :

$$S_n(r, k) = S_n(r, k-1) + \Delta S_n(r+k) \qquad (2.2)$$

with r and k positive integer variables, $k \geq 1$

Given two positive integer parameters μ, the maximum allowed for curvilinear integration amplitude, and kmx, maximum iterations number, the computation of the texture component $a_n(r)$ related to the orientation $\theta_n = n\Pi/4$ is given by the following algorithm :

```
IF there is a positive integer $k_f <$ kmx as $S_n(r, k_f-1) < \mu \leq S_n(r, k_f)$        (2.3)
    THEN $a_n(r) = k_f P(n)$
    ELSE (i.e. $\mu$ not reached in fewer than kmx steps)
    $a_n(r) =$ kmx $P(n)$
    ENDIF
```

With the aim of designing a processor dedicated to the M.C.I., certain factors have to be pointed out :

- In an area of an image, one ΔS_n sample is involved in the computation of several texture components related to neighbouring pixels. This feature may be taken advantage of in order to reduce I/O activity.
- Computations related to colinear scanning directions in a limited area are partially shared. In the special case of a connected pixels couple, the current curvilinear integrations only differ by the value of one ΔS_n sample. This is obviously a line to follow in order to reduce the complexity of the algorithm.
- The data dependence nature of the algorithm (the number of iterations involved in the computation of a texture component is not known beforehand) is an obstacle to highly concurrent implementation. The fine grain underlying parallelism involved in this low-level algorithm requires a tightly synchronous-scheduled design which conflicts with the stochastic feature of the execution scheme associated with the algorithm.

3 REDUCTION OF REDUNDANCY INVOLVED IN THE COMPUTATION PROCESS

3.1 SETTING UP OF A PREDICTIVE PROCEDURE

The term "predictive" is used by analogy with signal differential coding techniques, as far as the idea is concerned. As previously stated, the computations involved by colinear directions of two connected pixels are tightly linked. This is obvious in the figure 3.1.

Let $a_n(r) = k_f P(n)$ be a texture component related to pixel M. The curvilinear integration related to pixel M' in the same direction, may be easily bound to the previous one :

$$S_n(r, k_f) = \Delta S_n(r+1) + S_n(r+1, k_f-1) \qquad (3.1)$$

Moreover, the following relationship has been proved [OUVR.chap2.2], assuming k' is the iteration number involved in the computation of the texture component $a_n(r+1)$ related to pixel M' (i.e. $a_n(r+1) = k_f' P(n)$), then :

$$k_f' \geq k_f - 1 \qquad (3.2)$$

This leads to 'a computation saving process driven by data vectors. The relationship (3.2) garantees that no back-tracking has to be supported (i.e. a ΔS_n sample is always added one time and substracted one time), thus, outside the start-up phase, only two iterations (one to add, the other to substract ΔS_n) are required on average per texture component.

3.2 GOING EVEN FURTHER WITH THE PREDICTIVE PROCEDURE

Looking back at the figure 3.1 with cost-saving in mind leads to a consideration of the correlation between the computation related to θ_n direction and that related to the opposite direction. The latter will be designated as the θ_{n+4} direction. The relationship below is obvious

$$S_{n+4}(r+k, k) = S_n(r, k) \qquad (3.3)$$

Furthermore the following rule is demonstrated in [OUVR.chap2.3]:

```
Assuming $a_n(r) = k_f P(n)$,
    If there is an integer value $k^* \geq k_f-1$, such that
        $S_n(r+1, k^*) < \mu$ AND $k^* <$ kmx
    Then $a_{n+4}(r+k^*+1) = (k^*+1)P(n)$
```

This rule leads to a very interesting release of the predictive procedure. One can depict this as follows : When a θ_n component, say $a_n(r) = k_f P(n)$, is yielded, the predictive process goes on with the computation of $a_n(r+1)$ but directly at the (k_f-1) step. The stated rule steps in at this point : If none of the two thresholds μ and kmx is reached by $S_n(r+1, k_f-1)$, then $a_{n+4}(r+k_f) = k_f P(n)$. Moreover, the rule tells us that each further step which does not trigger the thresholds gives a new θ_{n+4} texture component.

From the point of view of global process scheduling, this leads to a very simple alternative behaviour including the I/O management :

EITHER

the process requests a new S_n sample => a $\#_{n+4}$ component is produced

OR

the process rejects an old S_n sample => a $\#_n$ component is produced

With this deeply restructured algorithm, the computation and I/O loads are, in the same proportion, dramatically reduced. But the data dependence of the process appears now more obvious in so far as k component computations are simultaneously affected.

4 TOWARDS AN EFFICIENT ARCHITECTURE FOR THE COST SAVING ALGORITHM IMPLEMENTATION

4.1 A CONVENIENT TYPE OF PARALLEL ARCHITECTURE

Like most signal processing algorithms, a pipeline approach is well suited for M.C.I. implementation. Indeed, the computation procedure involves data vector flows and may be easily split into a sequence of elementary tasks implemented by specific hardware.

Granted that a pipeline approach will act at the basic level of the hardware design, one has to consider also the appropriateness of a replicated structure. At the same basic level, this leads to systolic architecture, whereas at an upper level, it leads to SIMD/SPMD architecture (MIMD not suited for fine grain parallelism).

Systolic architecture is obviously the most adequate for a custom or semi-custom design (regular structure and scheduling, local communications, ...). But unfortunately this approach is quite inflexible and leads to serious obstacles with data dependent processing. Considering our algorithm from a systolic point of view, the two index function $S_n(r, k)$ looks like a possible candidate. But in this two dimension computation space, the only points involved in effective computations are mapped onto a path plotted during a data vector processing. Thus the computation process has to be considered as a one dimension one and, furthermore, the computation path is not known beforehand. Consequently the systolic approach is not available for our cost saving algorithm.

The SIMD approach (multiple identical pipes in our case) does not involve such a theoretical problem. But the M.C.I. is not computation demanding, therefore the "I/O bound" feature prevails. Furthermore, a pipeline implementation of the computation process will lead to an high throughput. In the case of a multiple processing unit design, the global bandwidth required for the processor memory will involve a costly and a complex design. Finally this approach does not have to be rejected a priori, but the tradeoff it brings about should be carefully examined.

4.2 THE PROCESSOR ARCHITECTURE

The data dependency feature of the algorithm led us to confine the critical computations tasks in a specific hardware section named *central processor* due to its situation inside the sequential global process. The less dependent tasks (i.e. "receive data and compute ΔS_n sample" on the one hand, and "compute texture component from k_f stream and transmit it", on the other hand) are supported by specific units as depicted in figure 4.1.

The latter shows a logical signal T by which the central processor schedules the whole system. This signal reflects the state of the threshold process (T is true when neither μ nor kmx are reached). For the preprocessor, "T is true" involves an active cycle (input and process data) when "T is false" requires an idle cycle (hold process state). It is quite different for the postprocessor because the latter is always busy. The state of T indicates the type of the texture component (a_n or a_{n+4}) whose computation has to be initiated. The T bit is tagged to the processed data and is then used by the memory controller in order to demultiplex the texture components stream.

From the point of view of pre- and postprocessor design, there is no theoretical limit to the degree of pipelining because no recurrent function is involved. That is why the paper will only focus on the central processor implementation.

4.3 CENTRAL PROCESSOR ARCHITECTURE AND BEHAVIOUR

Although the central processor supports recurrent computations and data dependent scheduling, it has to slow down the global pipelined process as little as possible. An extensive functional parallelism was required to achieve the aim of one texture component produced per machine cycle.

In order to equalize concurrent tasks execution time in a context of synchronous parallelism, the central processor architecture exhibits three functional subunits :

- the *data string management subunit* (implements a queued structure for ΔS_n samples)
- the *computation subunit* (computes S_n and updates k)
- the *threshold checking subunit*

Each of these is structured as a pipeline stage with only one clocked layer.

The critical point was to assume overlapped execution of mutually dependent tasks : the S_n and k computation step is subject to the threshold checking issue and vice versa. This problem was worked out by a dynamically configurable structure involving the duplica_ tion of algorithm objects and operators. The architecture of the central processor is depicted in figure 4.2.

The behaviour of the central processor is depicted in the *ASM chart* figure 4.3. As previously stated, each machine cycle yields a texture component (a_n if T false, a_{n+4} if T true).

4.4 PROCESSOR COST AND PERFORMANCE

On the basis of the *advanced low power schottky TTL standard devices*, the design of the processing unit involves fewer than 60 chips and leads to a throughput of 8 Mega texture components per second (125 ns cycle time). The required data memory bandwidth is about 12 Mbyte/s. It should be pointed out that the *data string management subunit* is the most time-critical function. One may expect that implementing this subunit by an ASIC leads to improvement in the performance by a factor two or three.

5 IMPLEMENTATION OF THE M.C.I. WITH A SYSTOLIC ARCHITECTURE

5.1 WHY A SYSTOLIC ARCHITECTURE ?

Taking into consideration the unique features of a systolic architecture which bring great benefits in hardware design [KUNG], we had to investigate a restructured algorithm consistent with this approach. The main benefits are integration facilities, extension facilities, I/O saving and high degree of parallelism achievable in a small package. Thus, this approach may be interesting even if the ad hoc algorithm is not the most cost effective one.

5.2 *SYSTOLIZING* THE BASIC SEQUENTIAL ALGORITHM

The technique we used with this in mind is described in [QUIN] and [ANDR]. It starts with a map of computation steps referenced by the index set involved in the latter. The next steps are :

- translate computation formula into a uniform recurrent equation system (i.e. position independent)
- define a convenient scheduling of computation points
- select an allocation function in order to map computation space onto a physical architecture.

The basic formulation parts of the M.C.I. which involve recurrent formulae are those previously stated (2.2) and, with the same data stream input, the curvilinear integration related to $\#_{n+4}$ which is given below :

$$S_{n+4}(r, k) = S_{n+4}(r, k-1) + \Delta S_n(r-k+1) \qquad (5.1)$$

The question of the data dependency of the computation process is not yet worrying because the components are computed by independent processes (only data input shared). It is first necessary to differentiate k as a computation space coordinate from k as the processed variable converging to k_f during its migration through this space. The latter is labelled k_f^*. An inhibition mechanism is then easy to implement in order to freeze its state when a threshold triggers till it leaves the systolic array (this is not necessary for S_n and S_{n+4}). A T tag bit, exactly similar to the one previously given, is used with this aim in view.

This problem solved, the translation of (2.2) and (5.1) into uniform recurrent equa_ tion systems is processed as follow. Considering first the case of (2.2), the $\Delta S_n(r+k)$ sample is used by all computation points (r', k') assuming $r'+k' = r+k$. This can be rewritten as $\Delta S_n(r, k) = \Delta S_n(r+p, k-p)$ with p an arbitrary integer value. Select the value of p leads to defining the ΔS_n path through the computation space. The value p=1 is obviously the most efficient and leads to the following equation system :

$$S_n(r, k) = S_n(r, k-1) + \Delta S_n(r+1, k-1)$$
$$\Delta S_n(r, k) = \Delta S_n(r+1, k-1)$$
with $0 \le r <$ (input data vector length)
and $0 < k \le kmx$

The associated bound conditions are $T(r,0)=TRUE$, $k_f^*(r,0)=0$, $S_n(r,0)=0$ and $\Delta S_n(r \ge 1,0)=\Delta S_n(r)$.

The value $k_f(r)$ related to the $a_n(r)$ component is achieved by extracting $k_f^*(r,k \ge kmx)$ from the computation space. The ASM chart figure 5.1 depicts the complete computation process executed at each point of the computation area.

The computation of S_{n+4} (5.1) leads to very similar results. The main difference lies in the ΔS_n path ($\Delta S_n(r, k) = \Delta S_n(r-1,k-1)$ in this case).

5.3 DESIGN OF THE RELATED SYSTOLIC ARCHITECTURE

We give a mere summary below of the functions chosen ; more details may be found in [OUVR.A].

Computation scheduling

The following functions define the time at which a computation is processed (for $\#_n$ and $\#_{n+4}$ respectively) :
$t_n(r, k) = r + 2k - 1$
$t_{n+4} = r + k - 1$

Computation allocation

The following function defines the processor cell which processes a computation point (idem for the two arrays).
$A(r, k) = k$

Main features of the design

- one dimension arrays, 32 (=kmx) cells each
- all cells busy at each cycle (i.e. 100% hardware utilization rate outside start-up phase)
- each array yields one texture component per machine cycle

The complete design of the systolic processing unit includes one preprocessor and two postprocessors (identical to those described in section 4) in addition to the two systolic arrays.

5.4 COST AND PERFORMANCE CONSIDERATIONS

The TTL device family is, in this case, not appropriate for cost evaluation of the hardware design. Therefore this is done in terms of elementary logical gates.

About 40,000 gates are involved in the design of this processing unit versus 2,000 for the previous solution. From the ASIC point of view, such a design does not involve great difficulties. In a "gate array" context, for instance with such a well mastered technology as CMOS-2μ, fewer than ten chips will be needed for this implementation.

From the performance point of view, it is obvious that the systolic unit offers an intrinsic throughput twice as great as the first proposed solution. Furthermore, another factor is perhaps more significant : the systolic unit does not involve critical macro function such as the *queued data string subunit* of the previous one. This allows the systolic design to get the best out of a leading technology. For instance, with today's *Advanced Schottky TTL* standard devices, an output rate of about 50 Mega texture components per second is achievable (involving 6 GigaOps/s).

6 CONCLUSION

In this paper two processor architectures dedicated to the Method of Curvilinear Integration implementation have been proposed. In the field of concurrent processes hardware implementation, the main axis is obviously the adequateness of algorithm and architecture. Therefore, it is, more often than not, more risky to present one architecture as the most efficient one.

In our case, features related to each design allow us to give some orientation : For the purpose of a high performance application (as TV real time texture analysis), the systolic approach is the only valid one. An accurate evaluation in the context of an AS-TTL design has shown a total time of 45 ms to analyze a 512 x 512 picture.

On the other hand, the first proposed design is characterized by its cost-saving : No sophisticated tools required for the design (standard IC) ; usual microcomputer RAM suitable for processor memory, providing a convenient DMA controller is designed. Such a low cost system would support the texture analysis of a 512 x 512 picture in as short a time as one second.

7 BIBLIOGRAPHY

[RONS] J. RONSIN, D. BARBA, S. RABOISSON: "Comparison between cooccurrence matrices, local histograms and curvilinear integration for texture characterization", Second International Symposium on Optical and Electro-optical applied Sciences and Engineering, pp.596-603, Dec. 1985.

[OUVR] G. OUVRADOU: "Conception d'un Processeur de Caractérisation de Texture", Doctorat Trait. Info., Ref. D87-9, INSA Rennes, juin 87.

[KUNG] H.T. KUNG: "Why Systolic Architectures ?", IEEE Transactions on Computers, pp.37-46, Jan. 1982.

[ANDR] F. ANDRE, P. FRISON, P. QUINTON: "Algorithmes Systoliques : de la théorie à la pratique", Publications internes IRISA No.196, Mars 1983.

[QUIN] P. QUINTON: "The Systematic Design of Systolic Arrays", Publications Internes de l'IRISA, No.193, Rennes, Avril 1983.

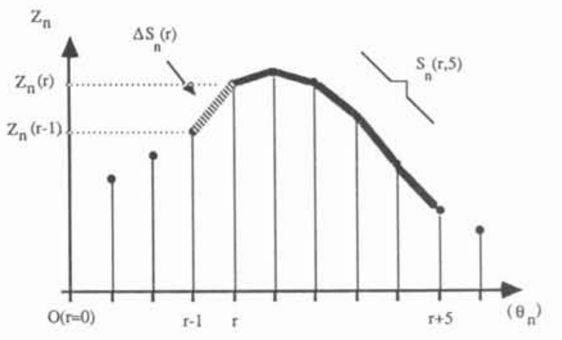

Figure 2.1

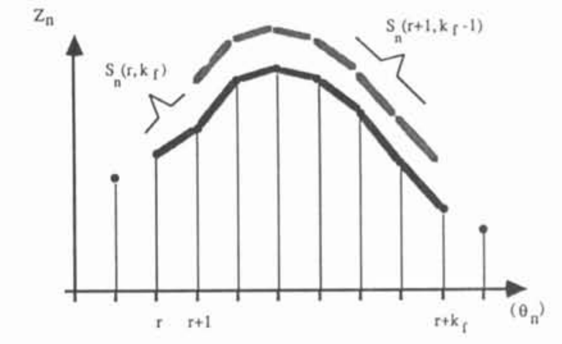

Figure 3.1

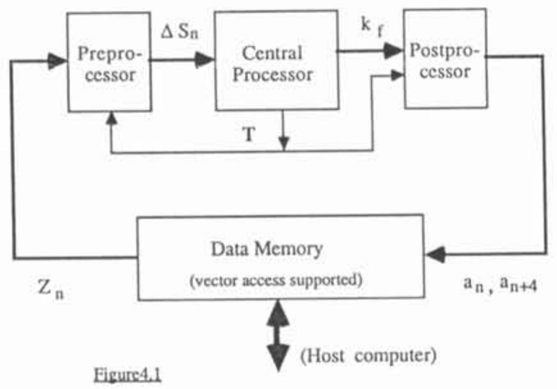

Figure4.1

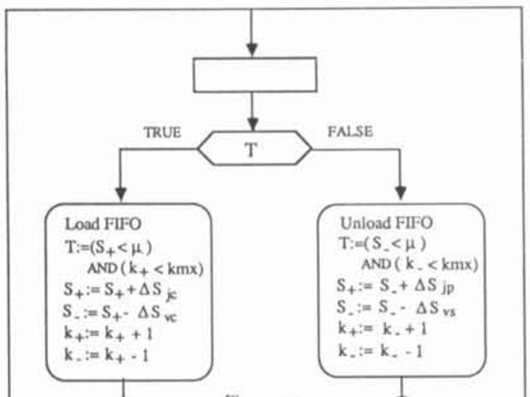

Figure 4.3

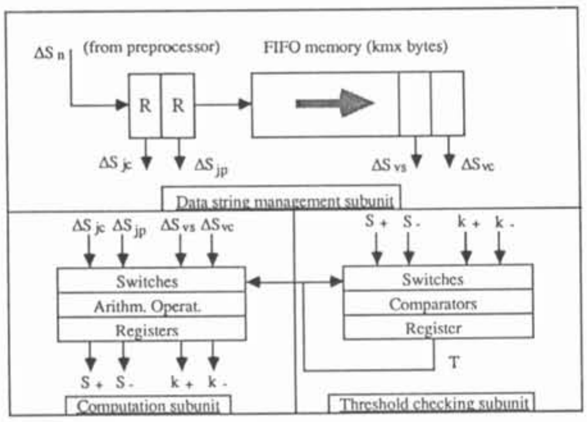

Figure4.2

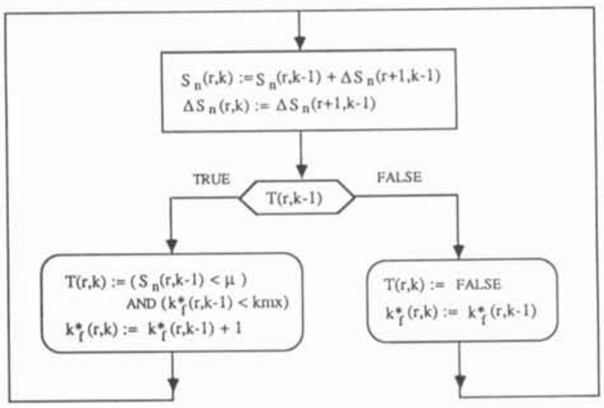
$S_n(r,k) := S_n(r,k-1) + \Delta S_n(r+1,k-1)$
$\Delta S_n(r,k) := \Delta S_n(r+1,k-1)$
TRUE
T(r,k-1)
FALSE
$T(r,k) := (S_n(r,k-1) < \mu)$ AND $(k^*_f(r,k-1) < kmx)$
$k^*_f(r,k) := k^*_f(r,k-1) + 1$
$T(r,k) :=$ FALSE
$k^*_f(r,k) := k^*_f(r,k-1)$

Figure 5.1