Abstract

A design methodology for the definition of parallel architectures for image processing is presented. The architectural design is based upon functional blocks, operating on limited regions of neighboring pixels (windows) in the image plane. This window-based computational granularity allows to cover a wide range of image processing algorithms, with particular reference to image enhancement and analysis.

The window-based formulation and decomposition of the algorithms is derived, allowing to identify the architectural characteristics of the window processor and of the required interconnection structures.

Evaluations of the architectural complexity guaranteed by this approach are given in terms of temporal and spatial requirements.

1 Introduction

Definition of architectures for computation of image processing algorithms can vary as the consideration of “hardwired” dedicated architectures to programmable structures, mainly depending on the requirements and the constraints of the envisioned application, but also on the considered phase of processing.

The low and medium levels of image processing, i.e. image enhancement and image analysis, are both characterized by a large amount of data to be processed (the computation is typically an image-to-image or an image-to-feature transformation), and by classes of algorithms that present a fine-medium grain parallelism. These two computational characteristics, together with possible real-time application environments, lead to consider the definition of dedicated parallel architectures, that execute the described processing with a high efficiency.

In this paper, we define a methodology for the architectural design of functional blocks for the realization of parallel dedicated structures, capable of executing a wide range of algorithms, belonging, in particular, to the low and medium phases of image processing.

This approach follows a design philosophy that can be referred to as the design of flexible dedicated architectures [2,3,4,5,6,7], in which a tradeoff is made between the high performances granted by parallel, one-algorithm, dedicated architectures and the possibility of an extension of the set of algorithms to be computed, without referring to a programmable implementation.

The definition of functional blocks and the identification of the structure of the architecture is done through the analysis and the decomposition of image processing algorithms.

In particular, the following characteristics have been considered as guidelines for the algorithmic analysis:

1. The nature of the computed result, i.e. the meaning and the possible usage of inputs and outputs.
2. The structure of the algorithm, i.e. the topology of the computational graph of the algorithm and the algebraic nature of the involved operators.
3. The phase of the elaboration chain. Substantially different algorithms may be cascaded to build a single higher-level processing.

Moreover, as a result of the followed approach, the proposed methodology allows to design dedicated architectures that fulfill the following requirements:

1. Realize a high modularity internally to the architecture itself. This means that the architectures present an internal functional modularity, based on an iterative usage of a restricted set of functional blocks.
2. Exhibit a high modularity transversally to the classes of algorithms. This means that architectures dedicated to substantially different algorithms can be designed by means of the same functional blocks.

2 Approach followed: the window-based processing

The proposed approach is based upon the identification of a set of lower-level algorithmic kernels, to which the computation of the different algorithms has to be reduced. These kernels represent the computational granularity of each algorithm. Algorithms need be reformulated, possibly in an iterative way, considering kernels as inner functionalities.

From an architectural point of view, these lower-level kernels represent the functional blocks in which the architecture is decomposed.

In order to derive this representation, we envision, as basic phases of the methodology proposed in this paper, the following points:

1. Capture the topological and algebraic similarities of image processing algorithms and identify common algorithmic substructures of potentially dishomogeneous classes of algorithms.
2. Define a mapping of algorithmic substructures onto a, possibly restricted, set of lower-level operators, to be themselves defined.
3. Analyze the complexity of the algorithm, with respect to the identified lower-level operators.
4. Design these operators by means of usual arithmetic and logic operators and connect them to build final architectures.

The first phase of this methodology consists in the identification and the definition of the computational granularity: to this purpose, the first choice that has to be done is the characterization of this granularity, with respect to the “region” of pixels in the \( M \times N \) image plane, which is assumed as the input of the computation.

We consider as regions of pixels windows of neighboring pixels. A window is a \( H \times K \) \((H, K \geq 1)\) square region, originating at the pixel \((i,j)\) in the image plane. The indexing of pixels inside the window is denoted as a displacement form the origin of the window itself.

Lower-level algorithmic kernels are algebraic operations and functions having windows as arguments. These operations and functions represent the computations performed by the functional blocks, i.e. by the window processors.

The chosen computational granularity allows to cover a wide range of image processing algorithms, with particular reference to image enhancement and image analysis. Examples of selected classes of algorithms [1], for which a window-based realization has been envisioned, are shown in tab. (1). After defining the computational granularity, the subsequent phase consists in the reduction of the global elaboration, over the whole image, to a formalization based on the iterative usage of window processors. The identification of the minimal number of window processors to be used, possibly by composition, gives eventually the structure of the architecture.

---

CH3001-91/0000/0243/$01.00 © 1991 IEEE

243
3 A methodology for the decomposition of image processing algorithms

In this section the formalism to decompose image processing algorithms in terms of window processing is presented. First we define the input values $P_i$ and the output values $P_o$ of the algorithms:

$$P_o = \{P_{i1}, P_{i2}, \ldots, P_{in}\}$$

is the result of the computation, belonging to a 2n-dimensional output space. $P_o$ may be a partial result, requiring further elaboration to obtain the desired final result.

Note that $P_i$ depends, in general, on the positions of the $n$ input windows, if their movements around the image plane are unbounded. If not so, the dimension of the output space might well be smaller than 2n.

The following are the domains on which the transformations, composing the image processing algorithms, operate:

- $I = [0, 2^d - 1]$ is the pixel domain (gray levels).
- $W = I^{H \times K}$ is the window domain.
- $\Omega = \{\omega_p\} 1 \leq p \leq m$ is the window operator domain. $\omega_p$ is a single window operator.

The general formula, to which algorithms for window-based processing are reducible, is:

$$P_o = \alpha(\bigcirc \bigtriangleup \bigtriangledown \gamma \beta \tau) \quad (1)$$

The transformations into which the algorithm is decomposed are detailed in the sequel. The description proceeds from right to left, thus following the order of application of the different transformations. Moreover, for each transformation the input and output domains are specified.

$$\delta : W \rightarrow W$$ are $n$ functions on a window to a window, operating "pixelwise".

$$\bigcirc : W^n \rightarrow W$$ is a composition law on $n$ windows to a window.

$$\bigtriangleup : \Omega \times W \rightarrow W$$ is an action law of a window operator on a window to a window.

$$\bigtriangledown : W^m \rightarrow W$$ are $m$ functions on a window to a window, operating "pixelwise".

$$\gamma, \beta, \tau : W \rightarrow I$$ are $m$ functions on a window to a pixel.

$\alpha : I \rightarrow I$ is a function on a pixel to a pixel.

The above operations and functions can be better specified:

In most cases the action law $\bigcirc : \Omega \times W \rightarrow W$ acts "pixelwise" on its window argument, i.e. the $m$ window operators $\omega_p \in \Omega$ are elaborations masks, to be applied onto the window argument of $\omega_p$.

In most cases the composition law $\bigcirc : W^n \rightarrow W$ composes "pixelwise" its window arguments.

Fig. (1) shows a pictorial representation of equation (1). Examples are, in general, image analysis elaborations. When parameter

$$m = |\Omega| = 1$$

Fig. (2) shows equation (2). Examples of algorithms are autocorrelation and, in general, statistic elaborations. When parameter

$$\eta = \alpha(\beta)$$

Table 1: Classes of image processing algorithms, with significant examples.
blocks of dedicated architectures
the third one related to the usage of these processors as functional
and the second ones referring to the structure of the window processor,
particular, three different characterizations can be derived: the first
allow to derive the characteristics of window-based architectures. In
eral, simple algorithms belonging to feature extraction elaborations,
and rank filtering. The single input window may collapse into a single
pixel, yielding a pure scalar elaboration. The application of the window-based
processing may result in the decomposition of the algorithm into a number
of iterations, or steps. The global elaboration is obtained by applying iteratively
this equation need not necessarily be simplified in the same way, when
plied in different iterations. For instance, statistical algorithms for
the computation of the moments require an iterated application of the
same equation (see section 6), while algorithms for the computation of
texture patterns, selecting a set of privileged directions, require the
ascended application of two different simplifications of equation (1).

Fig. (4) shows equation (4). Examples are linear filtering and, in
eral, simple algorithms belonging to feature extraction elaborations,
such as convolution and rank filtering. The single input window may
be one-dimensional \( H = 1 \); in this case, equation (4) can be further-
more simplified. Finally, the single input window may collapse into a single
pixel, yielding a pure scalar elaboration. The application of the window-based
processing may result in the decomposition of the algorithm into a number
of iterations, or steps. The global elaboration is obtained by applying iteratively
this equation need not necessarily be simplified in the same way, when
plied in different iterations. For instance, statistical algorithms for
the computation of the moments require an iterated application of the
same equation (see section 6), while algorithms for the computation of
texture patterns, selecting a set of privileged directions, require the
ascended application of two different simplifications of equation (1).

4 Window processor based architectures

The formalism and the algorithmic decomposition presented in section
3 allow to derive the characteristics of window-based architectures. In
particular, three different characterizations can be derived: the first
and the second ones referring to the structure of the window processor,
the third one related to the usage of these processors as functional
blocks of dedicated architectures.

1. Input/output characteristics of the window processor:
      (a) A window processor is applied to a \( n \times 1 \) \( H \times K \) regions of
          neighbouring pixels.
      (b) A window processor produces a single value as a result.
      (c) More generally, multiple outputs, possibly arranged in a
          window or a vector, could be produced, if equation (1) is
          truncated.

The three above characterizations determine the size of the input
data and the size and the indexing of output data.

2. Elementary operations and functions executed by the window processor.
      (a) Elementary operations have been found to be essentially
          arithmetic operations, plus some non linear functions, on
          natural numbers, and bitwise boolean operations, possibly
          composed with one another.
      (b) Input data may be subject to a multiple usage, in the case
          of algorithms which need several different window processors
          to be applied to generate the result (e.g. edge extraction
          and template matching algorithms).

3. Structural description of the window-based architecture. From
   the analysis of the selected algorithms, structural descriptions
   can be characterized as follows:
      (a) Dependency of the global output of the algorithm on the
          outputs of the window processors. This is related to the
          transformation performed by the algorithm on its input data:
              i. Image-to-image transformations: in this case, the out-
                 put of each window processor corresponds to one pixel
                 of the transformed image.
              ii. Image-to-feature transformations: in this case, the out-
                   puts of a window of window processors are iteratively
                   processed again by subsequent (possibly the same ones)
                   window processors.
      This dependency gives the number of window processors
to be cascaded and it defines the interconnection structure
among cascaded processors.
      (b) Dependency of the input windows, in the image plane. This
          dependency can be considered as the 
          \textit{decimation step} to be applied to the image plane, when moving
          the window around the plane itself, in order to define the set of in-
          puts of each window processor. For example, image or edge
          enhancement and filtering are characterized by a decima-
          tion step of a single pixel: in fact, the output value
          \( P_n(i,j) \) is generated starting from a window originating at pixel
          \( (i,j) \) in the image plane, while the output value
          \( P_n(i,j+1) \) is generated starting from a window originating at pixel
          \( (i,j+1) \). On the contrary, algorithms for the generation
          of pyramids of images, such as Laplacian or Gaussian pyra-
          mids \( \delta \), require a decimation step ranging from one pixel
          to the linear size of the window, depending on the desired
          resolution along the different planes of the pyramid. As
          a last example, statistical algorithms for the computation
          of global moments or spectral characterizations require, in
          the window-based approach, a decimation step equal to the
          linear size of the window.

The decimation step required by the algorithms allows to evaluate the number of window processors necessary to
compute the algorithms with the maximum parallelism, and it
defines the interconnection structure among processors
operating in parallel. If the maximum parallelism is not
exploited, the decimation step allows to define the struc-
ture and the functionality of an input module that provides
correct input data to the set of processors required by the
actual implementation by suitably scanning the image.

A possible general structure of the window processor, reflecting
the architectural requirements derived from point (1) and (2) above is
shown in fig. (5). The window processor has been organized in
three cascaded stages, namely:

1. The \textit{pre-processing} stage, that performs a pixel-wise transforma-
   tion on input pixels.
2. The \textit{window kernel} stage, that performs a transformation on \( n \geq 1 \) \( H \times K \) regions of
   neighbouring pixels, producing a single value
   as output, or also multiple outputs, if equation (1) is
   truncated.
3. The \textit{post-processing} stage, that generates the final result of the
   window processor, performing a one-to-one transformation.

The main reason of this organization is related to the fact that, given
an algorithm, the functions executed by the pre-processing and by the
post-processing stage can be computed before executing the algorithm.
5 Evaluation of the complexity of the architectures

To characterize the complexity of the window-based obtainable architectures, two main parameters have been computed, for each one of the four classes of algorithms shown in tab. (1). These two parameters give an evaluation of the complexity of the architectures in terms of temporal and spatial requirements, accordingly with the two measures generally adopted to define architectural complexity, i.e. the time and the area of a given circuit that implements the algorithm.

The two parameters give in fact a measure of the number of time steps necessary to perform the global computation required by the algorithms, and of the number of window processors required to compute the result. These measures allow to characterize the structural description of window-based architectures, as required by point (3) of the previous section, in terms of the number of window processors.

The two parameters have been evaluated under the hypothesis of maximum parallelism, i.e. it is assumed that the whole image is available and that the intrinsic parallelism of the window-based version of the algorithm is fully exploited at architectural level. Following this assumption, it results that:

1. The parameter $T$ represents the minimum number of time steps necessary to perform the computation.
2. The parameter $S$ represents the maximum number of window processors required to perform a given step of computation.
3. From the two parameters above, the total number $P$ of processors required can be computed.

The following further considerations can be derived:

1. In case of maximum parallelism, the value $T_{\text{top}}$, with $t_{\text{top}}$ representing the computation time associated to the slowest window processor, defines the latency of the architectures, i.e. the time interval between the input of the image and the production of the corresponding global result.
2. In case maximum parallelism, the maximum input frequency at image level, i.e. the maximum frequency at which a new image can be input into the architecture, is equal to $1/t_{\text{top}}$.
3. In case of minimum parallelism, i.e. if a single window processor is considered for the implementation of the architectures, the value $P_{\text{top}}$ represents the latency of the architectures, while the value $1/T_{\text{top}}$ represents the maximum allowed input frequency.

In tab. (3) the values for $T, S$ and $P$ are summarized for the four classes of algorithms considered in this paper.
Example 1: Template matching provides an example of the decomposition shown in Fig. (3). The neighbourhood of each pixel is matched with a fixed set of templates, i.e. window operators, and if a matching is found, then the pixel is considered. The general effect is to match edges and spots of the image, in order to perform later some sort of enhancement and/or extraction of features. Clearly outputs depend strongly on the number and on the geometry of templates. The specific instance of template matching that we consider can be described by the following equation:

$$P_{o}(i,j) = \sum_{p=1}^{m} \sum_{h,k=0}^{H-1} \sum_{j=0}^{K-1} (O(p,h,k) \otimes P_{l}(i+h, j+k))$$

where $P_{l}, P_{o} \in \mathcal{P}_{2}$, that is, the image is assumed to be BW. The equation represents the operation of matching the pixels of a $H \times K$ input window originating at pixel $(i,j)$ with a fixed template $O_{p}$, chosen in a set $\Omega$ of $m$ templates, by means of the logical conjunction ($\otimes$) of the symmetric difference ($\oplus$) of the corresponding pixels of the input window and of the template, and the operation of composing the matchings for all templates, by means of a symmetric difference ($\oplus$). As matching is exact, only a single template can be matched at a time. $P_{l}(i,j)$ is the output of the matching: if $P_{l}(i,j)$ is equal to 1 than one of the $m$ templates matches a $H \times K$ region of the image plane originating at pixel $(i,j)$. With reference to equation (3), the meanings of the various symbols for template matching, are (with $n = 1, m \geq 1$, $\forall P_{l} = P$):

$$o : \Omega \times W \rightarrow W \equiv \omega_{P} \oplus P \equiv (O(p,h,k) \otimes P_{l}(i+h, j+k))^{(H-1) \times (K-1)}$$

$$F : W \rightarrow I \equiv \sum_{h,k=0}^{H-1} \sum_{j=0}^{K-1} (O_{p}(h,k) \otimes P_{l}(i+h, j+k))$$

Note the following: $o$ computes the "pixelwise" symmetric difference of the template $O_{p}$ with a $H \times K$ neighbourhood at the pixel $(i,j)$; $F$ computes the logical conjunction of the pixels in a matched neighbourhood; $T$ computes the symmetric difference of a vector of $m$ values, corresponding to the $m$ matchings of the templates. With the decimation step $d = 1$, as it is in the general case, it results $T = 1$ and $S = M \times N$.

Example 2: Statistical algorithms for the computation of moments, such as mean value, deviation, energy, etc., provide examples where iteration can be applied: the image plane is partitioned into adjacent non-overlapping windows, and the moment is computed for each window. These moments are rearranged into a window, on which the computation is iterated, until a single value is obtained. We consider here the mean value:

$$P_{o} = \frac{1}{M \times N} \sum_{i,j=1}^{M \times N} P_{l}(i,j)$$

The iterative formulation (in the case of two iterations - i.e. two stages) is:

$$P_{o}^{2} = \frac{1}{M \times N} \sum_{h,k=1}^{H \times K} P_{o}^{1}(h,k)$$

$$P_{o}^{1}(h,k) = \frac{1}{H \times K} \sum_{h_{u}k_{u}=0}^{H-1}(H-1) \times (K-1) P(h_{u}H+k_{u}K+k_{u})$$

where $P_{o}^{2}$ is the final result, i.e. the mean value of the image, and $P_{o}^{1}(h,k)$ are the outputs of the processors of the first stage (the first ones to be applied to the input image). The outputs of these $M \times N$ processors are rearranged into a window, to be processed by the next single processor stage. Each processor belongs to the decomposition structure shown in Fig. (4), and processors at different levels perform the same function. Assuming $H = K$, $M = N$ and $M$ multiple of $H$, parameters $T$ and $S$ have the values $T = \log_{2}M$ and $S = \lfloor M/H \rfloor^{2}$, for the first stage, while $S = 1$ for the second (and last) stage.

7 Concluding remarks

The design methodology presented in this paper can be considered as an approach to high-level synthesis of highly modular application specific architectures. In fact, the most significant phases of this methodology are based on the decomposition of image processing algorithms into a restricted set of low-level operators, and on the identification of a mapping procedure allowing to obtain the final computation by means of the identified low-level operators. The chosen computational granularity, i.e. regions of neighbouring pixels (windows) in the image plane, allows to cover a number of classes of image processing algorithms, mainly referring to image enhancement and analysis. In the paper, the decomposition and the reformulation of algorithms in terms of window processing is shown. This decomposition allows to derive the architectural characteristics necessary to design the overall structure. In particular:

1. The architectural requirements of the window processors are identified, both in terms of input/output characteristics and of elementary operations to be performed.

2. The structural description of the window-based architectures is identified, giving the number of cascaded and/or parallelized processors necessary to implement the architecture itself.

As a result of the methodology proposed, the design of the architecture shows a high modularity:

1. A very restricted set of functional blocks need be implemented.

2. Given an algorithm, the corresponding window processor can be designed by suitably cascading these functional blocks.

3. Given an algorithm, the overall architecture is defined by a number of cascaded and/or parallelized window processors.

To properly evaluate the design effectiveness of the proposed approach, temporal and spatial complexity characteristics of the window-based obtainable architectures have been derived.

References


247