Full Alternate Jacobi Minimization and VLSI Derivation of Hardware for Motion Estimation

Étienne Mémin[1] and Tanguy Risset[2]

1 Valoria, Université de Bretagne Sud BP 1104, 56014 Vannes, France and

IRISA 35042 Rennes Cedex; memin@irisa.fr

2 IRISA/INRIA 35042 Rennes Cedex, France; risset@irisa.fr

Abstract

This paper proposes a design methodology for the semi-automatically derivation of hardware dedicated to a generic class of image analysis reconstruction problems. The study will focus in particular on the implications of the associated estimation algorithm and especially of the built-in underlying minimization on the hardware. A specific minimization strategy has been designed with a view to improving the efficiency of specialized hardware (in terms of clock cycle and surface area occupation). The VLSI derivation tool presented here is based on a high-level specification of the estimation updating rules defining the problem at hand. The complete derivation is illustrated on an edge-preserving optical-flow estimator.

Keywords: motion estimation, high level synthesis, VLSI, systolic array

1 Introduction

Many reconstruction problems in image analysis or computer vision are formulated as the minimization of an energy function. This objective function describes the interactions between the different variables modeling the image features of the problem under consideration. It is generally composed of two terms. The first one (the *data term*) expresses the global adequacy between the *observed data* and the *unknown variables* that have to be recovered. The second term (the *regularization term*) encodes some smooth prior on the desired solution. The essential role of the later is to tackle the inherent *ill posed* nature of the reconstruction problems at hand. The Tikhonov regularization approaches [PTK87] as well as the Markov Random Field-based image analysis [GG84] or the Minimum description length principle [Lec89] lead to the minimization of such global energy functions.

Simple regularization term penalizes solutions with high gradient. Although this kind of prior leads to a classical quadratic minimization problem, it has the disadvantage of smoothing out the spatial discontinuities (or edges) of the underlying real solution.

To circumvent this problem, edge-preserving regularization has been recently introduced in computer vision and image analysis [BR96, CBFAB97, GR92]. Edge-preserving regularization is characterized by the substitution of the classical quadratic norm of the Tikhonov regularization by a robust error norm function. This kind of function allows us to attenuate the large deviations occurring in the smoothness term. According to a reformulation result [CBFAB97, GM87] the robust regularization term may be rewritten as a sum of weighted quadratic potentials. The weights involved here are new *auxiliary variables* related to the spatial discontinuities of the problem.

The minimization of such resulting energy functions is usually carried out alternatively with respect to the unknowns and with respect to the discontinuity weights. However, the global minimization to be performed is often an intricate problem: the number of possible configurations is generally very large and the global energy function is usually non-convex. Computationally demanding stochastic relaxation algorithms are generally necessary to compute optimal solutions. Less CPU intensive iterative methods are usually preferred.

The major drawback of iterative methods is the amount of computations required to update the image. For real world applications the computation time quickly becomes prohibitive on workstations. As a result, these energy-based models may become conceivable and "realistic" in computer vision if and only if *(i)* a *real time* execution may be achieved and *(ii)* the machines on which those models are meant to be run are small enough to be used in embedded systems. Only hardware dedicated to a given problem is able to comply with such constraints.

The conception of dedicated chips is actually a very difficult problem requiring great experience and deep architectural and algorithmic knowledge to produce efficient taylor-made solutions. These inherent difficulties in fact arise from a dual problem analysis: the set up of an efficient algorithmic version having reasonable properties from the architectural implementation point of view, and then, the hardware design itself. Moreover these two stages are tightly interwoven. Whenever the chip concerned does not satisfy some prescribed constraints (execution time cycle, or surface area), the definition of the algorithm or the design choices must be further modified. A reliable tool to automatically (or semi-automatically) derive specialized hardware from algorithmic specifications attenuates these difficulties. It obviously greatly facilitates the VLSI design, but also allows the redefinition of the algorithm. Any change on the algorithm may easily be transferred on the hardware with the VLSI derivation tool.

Following these general remarks, the purpose of this paper, is to show that high level design methodologies can be successfully used to design efficient dedicated hardware for a generic class of energy based applications. We will first highlight the *regularity* and the *locality* of the minimization of standard energy-based model for image reconstruction problems. Then we will show how semi-automatic synthesis of specialized hardware may be conducted for this kind of methods with the MMAlpha tool.

The outline of the paper is the following. First, we present the general model for dense image reconstruction and the edge-preserving regularization principle. We also describe the way the minimization of this type of energy functions is usually conducted. In the second part of the paper, we show how interaction with the VLSI design has led us to consider a modified non linear minimization strategy. The resulting minimization algorithm is not only now well suited to VLSI implementation, but also has proved to be very efficient [MR98]. The last part of the paper is devoted to a high level design methodology illustrated on an optical flow estimation and practical evaluation of the resulting hardware.

2 Discontinuity-preserving regularization

The kind of reconstruction applications we focus on are modeled as the following minimization problem:

$$\hat{\boldsymbol{w}} = \arg\min \mathcal{H}(\boldsymbol{w}), \tag{1}$$

where $\boldsymbol{w} \triangleq \{\boldsymbol{w}_s, s \in S\}$ is the field of unknown variables indexed on the image lattice S (with $|S| = m \times m = n$). The field $\boldsymbol{w}$ may be either a p-vector field or a scalar field. Let $f \triangleq \{f_s, s \in S\}$ denote the observed data field. The general form for the energy function we will consider is the following:

$$\mathcal{H}(\boldsymbol{w}) = \frac{1}{2}\|f - \kappa\boldsymbol{w}\|^2 + \frac{\alpha}{2}\sum_{<s,r>\in\mathcal{C}} \rho(\|\boldsymbol{w}_s - \boldsymbol{w}_r\|), \tag{2}$$

where κ is a linear operator, α a positive number which balances the two terms of function $\mathcal{H}$, and $\mathcal{C}$ is the set of "cliques". A clique is a set of mutually neighboring sites for a given neighborhood system (4-neighborhood system throughout the paper). The function ρ is a standard robust M-estimator [Hub81] allowing to cope with the large deviations from the regularization term. These deviations particularly occur at spatial discontinuity locations.

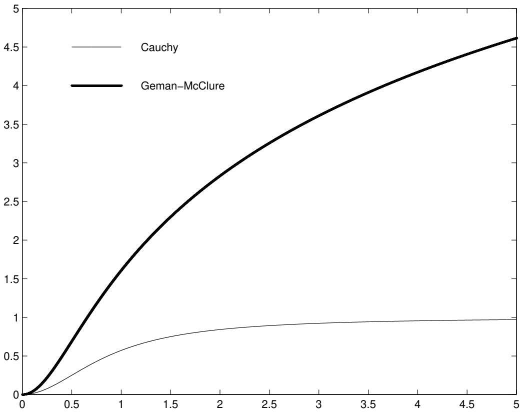
The archetype objective function we focus on is representative of most of the energy functions developed in reconstruction problems (such as image restoration [BR96, GM87], optical flow estimation [BR96], stereo-vision [RD96], computed tomography [Gre90]). In image restoration, f denotes the degraded observed intensity image and κ is the point spread function of the imaging system [GM87]. In computed tomography κ is the Radon transform, the data being photons counted over an array of detectors. In dense matching problem such as binocular stereo disparities estimation, dense depth map reconstruction [RD96] or optical flow estimation, the corresponding energy function may be written in this general form after a linearization of a non-linear data term arising from a constancy assumption. In optical-flow estimation this linearization leads to the well known *optical flow constraint equation* [HS81].

2.1 Iteratively Reweighted Least Squares Estimation

Let us now return to the robust estimator used in the model. Robust cost functions are continuous even functions, increasing on $\mathbb{R}^+$, and which penalize large "residual" values less drastically than quadratic functions do. This is usually achieved by letting their derivative, called *influence function* in robust statistics [Hub81], have finite limit at infinity (usually zero).

To give an insight into robust estimation as well as a practical way of handling it, it is fruitful transforming its use in terms of dual optimization problem involving *auxiliary variables* [BR96, CBFAB97, GR92]. To this end, we use the following reformulation result (see [CBFAB97] or [BR96] for a complete account): Let ρ be a real-valued continuously differentiable even function such that *(i)* ρ is increasing on $\mathbb{R}^+$; *(ii)* $\phi(v) \triangleq \rho(\sqrt{v})$ is *strictly concave* on $\mathbb{R}^+$; *(iii)* $\lim_{v\to+\infty} \phi'(v) = 0$, and *(iv)* $\tau \triangleq \lim_{v\to 0^+} \phi'(v) < +\infty$. Then there exists a strictly convex function ψ, continuously differentiable on $(0, 1]$, such that:

$$\forall u \in \mathbb{R}^+,\ \rho(u) = \min_{z\in(0,1]} \tau z u^2 + \psi(z). \tag{3}$$



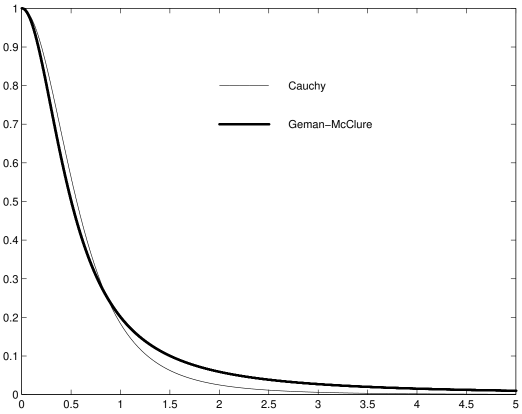


Figure 1: *Two robust estimators with the same inflexion point.* (a) *Cauchy's estimator* $\rho(u) = \log(1 + u^2/\sigma^2)$ *for* $\sigma^2 = 0.25$ *and Geman-MacClure's one* $\rho(u) = u^2/(\sigma^2 + u^2)$ *for* $\sigma^2 = 0.75$. (b) *Associated optimal weight function* $\hat{z}(u) = \sigma^2/(u^2 + \sigma^2)$ *and* $\hat{z}(u) = \sigma^4/(u^2 + \sigma^2)^2$.

This means that the graph of ρ is the inferior envelope of a family of parabola continuously indexed by $z \in (0, 1]$. The minimum in (3) is given in closed form by [BR96, CBFAB97]:

$$\arg\min_{z\in(0,1]} \tau z u^2 + \psi(z) = \frac{\rho'(u)}{2\tau u} = \sigma^2 \phi'(u^2), \tag{4}$$

where parameter $\sigma^2 \triangleq \tau^{-1}$ is analog to a variance ($\rho(u) \sim_0 u^2/\sigma^2$). Function ψ, which is never used in practice, is a strictly convex function ($\psi'' > 0$), [CBFAB97]. It is defined as:

$$\psi(z) \triangleq \phi \circ (\phi')^{-1}(\tau z) - \tau z (\phi')^{-1}(\tau z). \tag{5}$$

With such a cost function ρ, one can replace the multidimensional minimization in x of some $\sum_i \rho[g_i(x)]$ by the minimization in $(x, \{z_i\})$ of $\sum_i [\tau z_i g_i(x)^2 + \psi(z_i)]$ since both sums have the same global minimum in x. The extra variables z_is act as *adaptative weights* continuously lying in $(0, 1]$. Figure 1 shows two standard robust M-estimators with their respective weight functions.

In our case, the minimization of function $\mathcal{H}(\boldsymbol{w})$ (eqn. 1) is thus equivalent to the minimization of a new energy function $\mathcal{H}^\star(\boldsymbol{w}, \boldsymbol{\beta})$ involving discontinuity auxiliary variables (or weights) βs lying on the dual edge grid of S :

$$\mathcal{H}(\boldsymbol{w}) = \min_{\beta} \mathcal{H}^\star(\boldsymbol{w}, \boldsymbol{\beta}), \tag{6}$$

where

$$\mathcal{H}^\star(\boldsymbol{w}, \boldsymbol{\beta}) = \frac{1}{2}\|f - \kappa \boldsymbol{w}\|^2 + \frac{\alpha}{2} \sum_{<s,r>\in\mathcal{C}} \tau \beta_{sr} \|\boldsymbol{w}_s - \boldsymbol{w}_r\|^2 + \psi(\beta_{sr}). \tag{7}$$

In matrix notation this function may be written as:

$$\mathcal{H}^\star(\boldsymbol{w},\boldsymbol{\beta}) = \frac{1}{2}\boldsymbol{w}^T A_{\boldsymbol{\beta}}\boldsymbol{w} - \boldsymbol{b}^T\boldsymbol{w} + c_{\boldsymbol{\beta}}, \tag{8}$$

where $\boldsymbol{b} \triangleq \kappa^T f$, $c_{\boldsymbol{\beta}} \triangleq \frac{1}{2}\left(\|f\|^2 + \alpha\sum_{<s,r>\in\mathcal{C}}\psi(\beta_{sr})\right)$ and $A_{\boldsymbol{\beta}}$ is a symmetric positive definite matrix such that $A_{\boldsymbol{\beta}} = \kappa^T\kappa + \alpha\tau L_{\boldsymbol{\beta}}$, and L is a sparse matrix defined as (ν denoting the neighborhood system):

$$\begin{cases} L_{\boldsymbol{\beta}_{sr}} = -\beta_{sr}\mathbb{I}_p & \text{if } s\in\ \nu(r), 0 \text{ otherwise} \\ L_{\boldsymbol{\beta}_{ss}} = \sum_{j\in\nu(s)}\beta_{sr}\mathbb{I}_p & \end{cases} \tag{9}$$

The minimization of the new compound function is usually led alternatively with respect to the unknown $\boldsymbol{w}$ and to the weights $\boldsymbol{\beta}$. First, the weights being fixed, the dual energy function $\mathcal{H}^\star$ is quadratic in $\boldsymbol{w}$ (eqn. 8). One has thus to face a standard weighted least squares problem equivalent to the resolution of the (sparse) linear system $A_{\boldsymbol{\beta}}\boldsymbol{w} = \boldsymbol{b}$. Due to its dimension this linear system is solved by standard iterative methods (mainly *Jacobi* or *Gauss-Seidel*). For a fixed value of $\boldsymbol{w}$, following the previous theorem, the weights minimizing the energy function are computed explicitly from equation (4). The whole alternate procedure constitutes an *iteratively reweighted least squares* estimation (IRLS) [HW77]; it may be summarized by the following couple of equations:

$$\begin{cases} \boldsymbol{w}^{j+1} = & \arg\min_{\boldsymbol{w}}(\mathcal{H}^\star(\boldsymbol{w},\boldsymbol{\beta}^j)) \\ \boldsymbol{\beta}^{j+1} = & \arg\min_{\boldsymbol{\beta}}(\mathcal{H}^\star(\boldsymbol{w}^{j+1},\boldsymbol{\beta})) = \left[\ldots \frac{\rho'(\|\boldsymbol{w}_s^{j+1}-\boldsymbol{w}_r^{j+1}\|)}{2\tau\|\boldsymbol{w}_s^{j+1}-\boldsymbol{w}_r^{j+1}\|}\ldots\right]^T, \end{cases} \tag{10}$$

where j stands for the iteration subscript, $\boldsymbol{w}^j \triangleq [\boldsymbol{w}^j_{s_1}\ldots\boldsymbol{w}^j_{s_n}]^T$ and $\boldsymbol{\beta}^j$ is a q-components vector[1], $\boldsymbol{\beta}^j \triangleq [\beta^j_{c_1}\ldots\beta^j_{c_q}]^T$ where $c_\bullet$ denotes two-site cliques. The convergence proof of this technique toward a unique minima (assuming the convexity of $\mathcal{H}(\boldsymbol{w})$) may be found in [CBFAB97, Hub81].

The value of $\boldsymbol{\beta}^{j+1}$ can be computed explicitly (second equation of (10)), but $\boldsymbol{w}^{j+1}$ must be estimated with an iterative method (Jacobi or Gauss-Seidel), hence we introduce another index k which represents the stage of the iterative method. In the Jacobi case, the complete description of the algorithm is then:

$$\begin{cases} \boldsymbol{w}^j_{k+1} = & \boldsymbol{w}^j_k + \boldsymbol{p}^j_k \quad \text{with } \boldsymbol{p}^j_k = D^{-1}_{\boldsymbol{\beta}}(\mathbf{b} - A_{\boldsymbol{\beta}}\boldsymbol{w}^j_k), \\ \boldsymbol{w}^{j+1} = & \lim_{k\to\infty}\boldsymbol{w}^j_k \end{cases} \tag{11}$$

where $D_{\boldsymbol{\beta}}$ is the diagonal part of $A_{\boldsymbol{\beta}}$ (or block diagonal in case $\boldsymbol{w}_s$ is not scalar).

2.2 Definition of Full Alternate Jacobi Minimization

This section explains the influence of the hardware design process on the original algorithm specification.

In sequential implementations, the Gauss-Seidel method is usually preferred because it converges faster than the Jacobi method. But when we target implementation on massively parallel architectures, this advantage is counterbalanced by the fact that all

[1] $q = |\mathcal{C}| = 2m(m-1)$ in case of a 4-neighborhood.

the computations of one Gauss-Seidel iteration cannot be executed in a fully parallel way. Indeed, computations realized at stage k use both results of current stage k and results of stage $k-1$. This induces a more complex control for the Gauss-Seidel method implementation. Moreover, the convergence speed will no more be a crucial issue since the VLSI implementation will provide very fast execution. So, the complexity and cost of the design will be far more critical. These considerations led us to select the Jacobi iterative method.

From a VLSI implementation point of view iteratively reweighted least squares minimization also has some limitations. First, a control procedure is needed to handle the switch between the weight computing steps and the weighted least squares minimization. Such a control procedure makes processors more complex and induces an increase of the chip size. The second reason is even more important: practical experiments demonstrate that, as the iteratively reweighted least squares minimization proceeds, the number of iterations necessary to compute the weighted least squares minimization decreases at each step (the Jacobi algorithms computing $\boldsymbol{w}^j$ in (11) converges faster and faster as j increases). But this dynamic control is very difficult to implement efficiently in hardware, hence each Jacobi process will generally be run for a fixed number of iteration. This may therefore produce very inefficient computations (the processors will spend most of their time computing identities).

The idea that arises when deriving the hardware with the methodology described in section 3 is the following: given that the hardware is required to compute the weights, it is better to update the weights at each iteration of the algorithm than to compute them once the Jacobi algorithm has converged. The clock cycle of the circuit depends on the critical path of the hardware design. This critical path will be the same whether we compute the weight at every iteration or in a more episodic way. The desire of obtaining a regular architecture with a simple repetitive behavior was also an argument for this decision.

We propose here a modification of the iteratively reweighted minimization which yields a "fully" local and regular minimization scheme. This property is essential and greatly facilitates the architectural synthesis.

The proposed modification of the IRLS (10) consists in interweaving the two alternate steps in a couple of iterative equations rather than in having an alternation between the update of the weights and the iterative resolution of linear system. Here, the resolution of the linear system at fixed weights is replaced by one Jacobi iteration. The weights are then updated considering this current vector. The resulting Full Alternate Jacobi Method (FAJM) may be described by the following system:

$$\left\{ \begin{array}{lll} \boldsymbol{w}^{k+1} = & \boldsymbol{w}^k + \boldsymbol{p}^k & \text{with } p_k = D_{\boldsymbol{\beta}_k}^{-1}(\underbrace{\boldsymbol{b} - A_{\boldsymbol{\beta}_k}\boldsymbol{w}^k}_{\mathbf{r}^k}), \\ \boldsymbol{\beta}^{k+1} = & \arg\min\limits_{\beta}(\mathcal{H}^\star(\boldsymbol{w}^{k+1}, \boldsymbol{\beta})) = & [\ldots \frac{\rho'(\|\boldsymbol{w}_s^{k+1} - \boldsymbol{w}_r^{k+1}\|)}{2\tau\|\boldsymbol{w}_s^{k+1} - \boldsymbol{w}_r^{k+1}\|} \ldots]^T. \end{array} \right. , \qquad (12)$$

the expression of the p-components vector $\boldsymbol{\beta}^k$ being computed from (4). Vector $\mathbf{r}^k$ is the residual and $\mathbf{p}^k$ is the gain. The convergence proof of this minimization strategy as well as experiments on convergence speed can be found in [MR98].

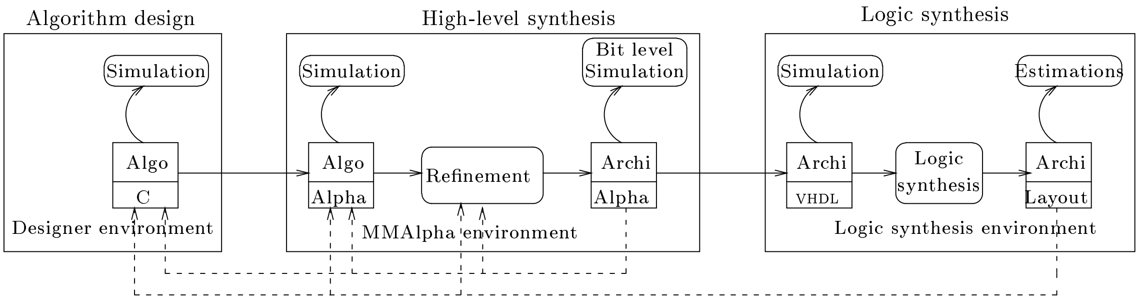


Figure 2: *The synopsis of a global design. The languages of the different specifications are indicated. Dashed lines represent possible backward annotations of specifications from lower level performance estimation.*

3 Derivation of hardware for energy based applications

In this section, we detail how the synthesis of dedicated hardware can be done from high level specification of algorithms. The experiments have been conducted on the particular application of motion estimation, but we try to give a view of the process that is as global as possible.

3.1 Methodology

Hardware specification is very different from algorithm specification. Firstly, the scientific background in hardware synthesis community and image processing algorithm designers are generally very different, hence communication between these two communities is difficult, whenever it happens. Secondly, slight modifications of an original algorithmic specification may or may not imply very important modifications on hardware, increasing or decreasing the complexity of the circuit generated. Finally, since these two specifications are expressed in very different languages (like for instance, VHDL and C), backward annotation, from a non-satisfactory hardware to the original algorithm is complex, it is indeed almost impossible to manage automatically (or at least semi-automatically) such backtracks. Furthermore, the inherent translation between different formats is a potential source of errors which are awkward to check. This leads most of the time to the following strategy: fixing definitively the algorithm before deriving hardware and deriving one single kind of hardware. All these difficulties increase the cost and the time to market of a dedicated hardware. Ideally, the design process should be much more flexible. To that end and in order to circumvent the aforementioned design problems, we propose here an approach based on a high level synthesis.

More precisely, our derivation process is based on the Alpha language [Wil94, Mau89]. Alpha is a functional language used in a program transformation environment called MMAlpha. Manipulating algorithms in Alpha has the following advantages:

- It can describe high level functional specifications of algorithms as well as low level descriptions of hardware. Hence it provides a common useful platform for image processing engineers and hardware synthesis engineers.

- The translation from the original high level specification into the final hardware specification is done by successive refinements of the original specification. The Alpha environment provides a set of transformations which are proved to be correct (thus no *hand made* transformation is necessary). The language allows strong static checks at each stage. The successive specifications may also be simulated at each stage.

- Once the basic design flow has been set for a particular algorithm, it can very easily be modified in order to explore the different possibilities in the design space. The original specification can also be modified according to some results of the design process.

Figure 2 represents the overall architecture of a complete design flow with a tool like MMAlpha. The design methodology is based on the systolic design methodology [QR89, Mol82]. The designer environment is chosen by the algorithm designer. Usually algorithms are simulated in imperative languages (like C of Fortran). The translation to the Alpha format corresponds to an imperative to functional code translation [Hel94]. The refinement in the MMAlpha environment is described in section 3.2, the translation to VHDL (hardware description language) is done automatically. The last part of the design is the logic synthesis which is usually performed with commercial tools.

3.2 Example of design methodology

We have chosen to illustrate the design methodology on the optic-flow estimation, section 3.3 gives practical results of the synthesis.

The generic mathematical method based on full alternate Jacobi minimization for solving discontinuity energy-based applications has been presented in equation (12). The first step of the design methodology is to translate the initial specifications (eqn (12)) in a functional formalism (in our case, recurrence equations [QR89]). Considering that the original image, f, is indexed with two indices i, j, and that $f_x(i,j), f_y(i,j)$ and $f_t(i,j)$ are the derivatives along x axis, y axis and time. The algorithm can be expressed, according to a 4-neighborhood system (North, South, East, West), with the following recurrence equations: (n is the index of the Jacobi iteration):

$$
\begin{array}{l}
\forall\ (i,j,n) \in \{i,j,n | 1 \leq i,j \leq m; 1 \leq n \leq p\} \\
\left\{
\begin{aligned}
\boldsymbol{w}(i,j,n) =&\ \bar{\boldsymbol{w}}(i,j,n) - P(i,j,n)\nabla f(i,j) \\
P(i,j,n) =&\ \frac{\nabla f(i,j) \cdot \bar{\boldsymbol{w}}(i,j,n) + f_t(i,j)}{\alpha\left(\beta_w(i,j,n) + \beta_e(i,j,n) + \beta_n(i,j,n) + \beta_s(i,j,n)\right) + \|\nabla f(i,j)\|^2} \\
\bar{\boldsymbol{w}}(i,j,n) =& [\beta_w(i,j,n)\boldsymbol{w}(i,j-1,n-1) + \beta_e(i,j,n)\boldsymbol{w}(i,j+1,n-1) + \\
& \beta_n(i,j,n-1)\boldsymbol{w}(i-1,j,n) + \beta_s(i,j,n)\boldsymbol{w}(i+1,j,n-1)]/[\beta_e + \beta_w + \beta_s + \beta_n] \\
\beta_w(i,j,n) =&\ \frac{1}{\tau}\phi'\left[\|\boldsymbol{w}(i,j,n-1) - \boldsymbol{w}(i,j-1,n-1)\|\right]
\end{aligned}
\right.
\end{array}
\tag{13}
$$

For sake of concision only "west" weight definition has been given (β_w), similar equations occur for the three others (β_e, β_n and β_s involving respectively $\boldsymbol{w}(i,j+1,n-1), \boldsymbol{w}(i-1,j,n-1))$ and $\boldsymbol{w}(i+1,j,n-1))$. For the same reason, equations corresponding to the initialization of each signal have not been included. The above specification can be

```
system mouv :  {N,M,P | 3<=N; 3<=M; 3<=P}
               (fx : {i,j | 1<=i<=N; 1<=j<=M} of real;
                fy : {i,j | 1<=i<=N; 1<=j<=M} of real;
                ft : {i,j | 1<=i<=N; 1<=j<=M} of real;
                coef :  real)
       returns (u : {i,j | 1<=i<=N; 1<=j<=M} of real;
                v : {i,j | 1<=i<=N; 1<=j<=M} of real);
var
  ut,vt: {i,j,n | 0<=i<=N+1; 0<=j<=M+1; 1<=n<=P} of real;
  ub,vb,PP : {i,j,n | 1<=i<=N; 1<=j<=M; 1<=n<=P} of real;
let
  ub[i,j,n] = (ut[i-1,j,n] + ut[i+1,j,n] + ut[i,j-1,n] + ut[i,j+1,n]) / 4[];
  vb[i,j,n] = (vt[i-1,j,n] + vt[i+1,j,n] + vt[i,j-1,n] + vt[i,j+1,n]) / 4[];
  PP[i,j,n] = (fx[i,j] * ub[i,j,n] + fy[i,j] * vb[i,j,n] + ft[i,j]) /
               (coef[] + fx[i,j] * fx[i,j] + fy[i,j] * fy[i,j]);
  ut[i,j,n] =
      case
        {| n=1} | {| i=0} | {| i=N+1} | {| j=0} | {| j=M+1} : 0[];
        {| 1<=i<=N; 1<=j<=M; 2<=n} : ub[i,j,n-1] - fx[i,j] * PP[i,j,n-1];
      esac;
  vt[i,j,n] =
      case
        {| n=1} | {| i=0} | {| i=N+1} | {| j=0} | {| j=M+1} : 0[];
        {| 1<=i<=N; 1<=j<=M; 2<=n} : vb[i,j,n-1] - fy[i,j] * PP[i,j,n-1];
      esac;
  u[i,j] = ut[i,j,P];
  v[i,j] = vt[i,j,P];
tel;
```

Figure 3: *Alpha code for the motion estimation from specification of eq. (13). This program corresponds to* `P` *iterations of the Jacobi method on an* `N`$\times$`M` *image. u and v are the two components of the vector $\boldsymbol{w}$ of equation 13.*

syntactically transformed into Alpha code which is represented on figure 3 in the case of constant unitary weights (Jacobi method).

The initial step of the design methodology (referred in the literature as *uniformization* or *localization* [DQ88]) transforms all broadcasts into a pipeline of value in order to obtain a locally connected architecture. Here for instance, we will need to pipeline the image variables, `fx[i,j]`, `fy[i,j]`, `ft[i,j]`, to all computation points `(i,j,n)`. This processes replaces, for instance, in the program of figure 3, `fx[i,j]`, by `Pipefx[i,j,n]` everywhere it is used and adds the following pipeline equation:

```
 Pipefx[i,j,n] =  case
        {| n=1; } : fx[i,j];
        {| 2<=n<=P; } : Pipefx[i,j,n-1];
      esac;
```

The resulting program is now uniform (every dependence vector is constant). There is no more data broadcasts (except for the constant `coef` which can does not change through the algorithm), the resulting architecture involves only local communications between neighboring processing elements. The next steps, namely, *scheduling* and *allocation*, constitute the parallelization stages. They govern intrinsically what kind of architectures we are finally going to deal with.

Scheduling of uniform recurrence equations has been widely studied (see [DR94] for example); it gives an execution date for all the computations. An automatic schedule procedure is provided in MMAlpha, a global clock is assumed and the execution date is given by a clock counter. The designer can specify which signal he wants to store (in register) in one iteration computation (i.e. in the computation of `ut[i,j,n]` from `ut[i,j,n-1]`). The simplest choice (which leads to the longest clock cycle) is to set only one register on the path between `ut` and `ub` (and the same for `v` of course. This yields the following schedule: all local variable at point `(i,j,n)` are scheduled at clock tick `n`, the results `u, v` are available at clock tick `1+P`. the variables `ub[i,j,n], vb[i,j,n], PP[i,j,n]` computed at step `n` are stored in a register until they is used (at step `n+1`).

The designer has now to specify the allocation function. This function is usually a projection of the dependence graph. Here, the trivial projection is $(i, j, n) \rightarrow (i, j)$ which means that each virtual processor is in charge of one pixel of the motion image. In Alpha, once the schedule and projection are chosen, one can express the computation in a basis such that the first index represents the time and the other the processors, this is very convenient for a structural interpretation of the code. The new indices of the resulting program are now denoted: `t,p1,p2`.

At this point, the designer can check whether or not the image is present in the array at the beginning (and whether or not the result should be output from the array). Here, the new equation defining `Pipefx` (i.e. the image used at each step) is:

```
Pipefx[t,p1,p2] =  case
       {| t=1; } : fx[p1,p2];
       {| 2<=t<=P; } : Pipefx[t-1,p1,p2];
      esac;
```

If we decide that the initial image `fx[i,j]` should arrive on the first column of processors (to respect I/O hardware constraint), then another pipelining transformation can be performed. The new equations for `Pipef` are:

```
Pipefx[t,p1,p2] =                  PipeIOfx[t,p1,p2] =
  case                                   case
    {| t=1} : PipeIOfx[t,p1,p2];           {| p1=0} : f[1-t,p2];
    {| 2<=t<=P} : Pipefx[t-1,p1,p2];       {| 1<=p1} : PipeIOfx[t-1,p1-1,p2];
  esac;                                  esac;
```

The array showed on the left of figure 4 results then from a possible projection. From that point onward, we have a representation of the array (size, shape) and a functional behavior of each processor of the array (i.e., instruction executed at each time step of the global clock). In the example of figure 4, data enter the array from the left and results are flushed out on the right.

The main difficulty in the low level transformations is the control generation. Indeed, all the controls realized with operators of high level languages (such as the restriction to domains on the Alpha program of figure 3) must be realized in hardware. Fortunately, due

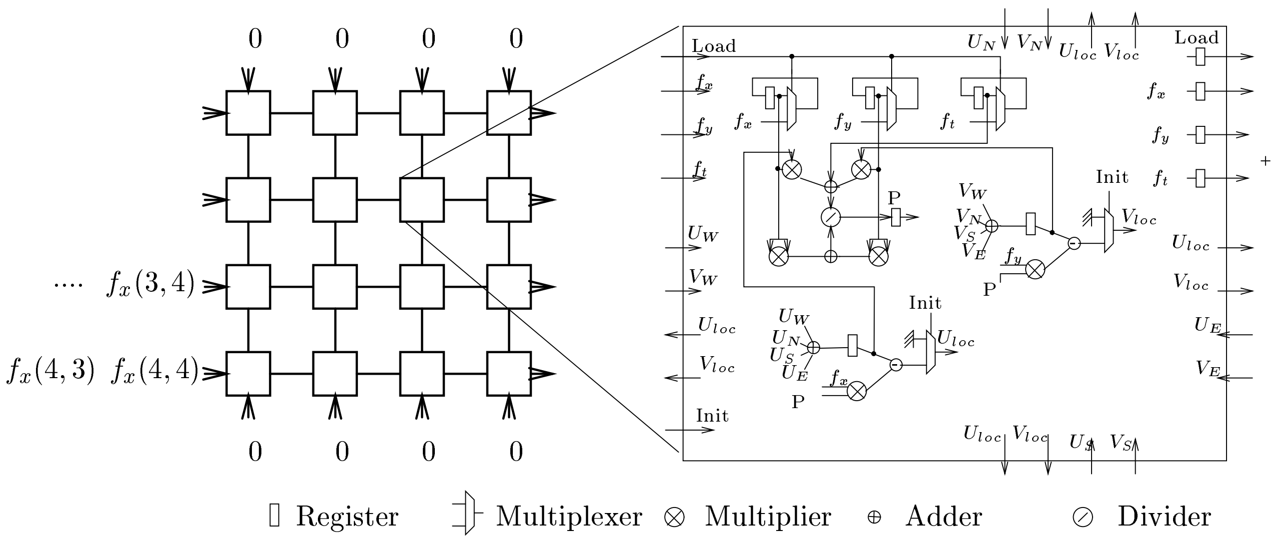


Figure 4: *Processor array (Jacobi method) for motion estimation. Original derivatives (fx, fy, ft) are input on the left of the array and the result is output on the right of the array. The schematic of one cell is detailed on the right. load and init are control signals. The hardware has been synthesized for optimum clock cycle; with greater clock cycles, some resources could be reduced (i.e. share operators)*

to the computation regularity of uniform program in Alpha, the control signal generation can be done automatically. In the current example of figure 4, the *load* control signal is broadcasted (all the processors start the computation simultaneously) but the loading of the images is pipelined from left to right.

After the control generation step, an Alpha program which can be directly interpreted as an architecture is obtained. One possible realization here is illustrated figure 4. In this implementation, two control signals have been generated: *load* indicates the loading of the derivatives in a local register and *init* indicates the beginning of computation. The data (derivatives) is pipelined through the array but the control signals are broadcasted (all processor start their computations together).

All these steps have been executed by MMAlpha functions controlled by the designer with a simple command line. The last stage in MMAlpha is the adjustment of the bit sizes of each signal. A bit level simulation can be done in the MMAlpha environment but this has to be tightly supervised by the user because it involves a subjective appreciation of the results.

The translation to synthetizable VHDL from this specification is automatic with MMAlpha and the synthesis can continue with commercial tools. As mentioned earlier, this design process can be easily modified in order to obtain other designs (for instance, pipelining control signals instead of broadcasting them). Depending on the synthesis results (here, presented in section 3.3), we may want to partition the array [MF86]. This transformation is not currently implemented in MMAlpha, but will soon be. Partitioning is useful, because, as exemplified in section 3.3, this design methodology often exploits "too much" parallelism (for the applications targeted here of course).

design characteristics	area of 1 cell (mm^2)	Clock cycle ($10^{-9}s$)	time for convergence (s)
Jacobi (8 bits)	0.4 mm^2	37 ns	9.2 10^{-6}s
Jacobi (14 bits)	1.4 mm^2	56 ns	14 10^{-6}s
FAJM (8 bits)	9.52 mm^2	89 ns	22 10^{-6}
FAJM (14 bits)	24.79 mm^2	117 ns	29 10^{-6}s

Table 1: *Experimental results of hardware estimation for one cell of optical flow estimation architecture. The robust estimator used here is* $\rho(x) = \frac{x^2}{x^2+\sigma^2}$. *Convergence time is evaluated with 250 iterations, the technology used is* $\lambda = 0.6\mu m$

3.3 Design process results

Architectures were synthesized for the motion estimation problem. A bit level simulation in MMAlpha was performed and an appropriate bit width was chosen for each signal. The Jacobi and the FAJM methods have been implemented. For sake of computation simplicity the Cauchy M-estimator was chosen as the robust cost function. Bit-level simulations showed that 14-bit wide signals were sufficient to provide good results.

Logic synthesis was carried out (on the architecture roughly depicted in figure 4) for the Jacobi and FAJM method, with 8-bit and 14-bit wide signals. For this architecture, we tried to maximize parallelism, and we did not suppose that the initial data was in the array (this may occur if we include sensors). Table 1 shows the estimated clock cycle of the architecture and the surface area for one cell (i.e. one pixel). These estimations were carried out using the Compass logic design tool on the VHDL code obtained with the MMAlpha translator. No divider was provided in the Compass library and was therefore replaced by multipliers during the evaluations. All operators were fixed point operators.

The number of iterations chosen (250) was far above what was needed for convergence, but was set so to have enough security with complex images. These results give an idea of the size of our array. Table 1 shows the estimated clock cycle of the architecture and of one cell area. A simple Jacobi relaxation for a 64×64 image would take a surface of 7.5 cm $\times$ 7.5 cm (with a motion estimation each 10^{-5} second). It must also be born in mind that technology is constantly increasing integration and that the current design has not been optimized (by hand) at all.

4 Conclusion

We proposed a design methodology for hardware synthesis of discontinuity preserving energy-based applications. This technique can be applied in various areas (image restoration, optical flow estimation, stereo-vision, ...). To this end, we proposed a new formulation of the usual minimization technique (namely Iteratively Reweighted Least Squares Estimation) for easier and more efficient hardware implementation. The convergence of the new non-linear method (FAJM) has studied in [MR98] and the experimental comparison between this method and standard minimization techniques in that context have demonstrated its efficiency and relevance. In this paper, we propose a complete and practical evaluation of the obtained hardware (in terms of area and clock cycle) designed through a high level VLSI derivation tool.

The complexity of the architecture generated is inherent to the class of algorithms chosen in this paper. However, recent research in computer vision tend to focus more and more on this kind of algorithm [tHR94], justifying the choice of these hardware implementations. Finally, the derivation technique that we used for VLSI design can also be used for FPGA design. Indeed the regularity of the design and the automation of the process are very useful if we target a fast implementation on FPGA.

The idea of integrating a processor *behind* a sensor has been studied for a while [TM86], and the concept of *smart censors* [FA92] and *Active Pixel Sensor* [MKF94] led to numerous achievements. Bi-dimensional arrays of such censors have already been implemented [AB95]. However, because of current technology restriction, these smart censors are often implemented with full custom design (often in analog technology), which is time consuming and error prone. We believe that the high level design methodologies that have been developed for digital technology enables the achievements of more secure design. Moreover, in a few years, when the technology allow greater integration, the power consumption and price will be advantages for digital technology.

acknowledgments

The authors gratefully acknowledge Dr. Patrick Pérez for his comments and numerous fruitful discussions.

References

[AB95] A.G. Andreou and K.A. Boahen. A 590,000 transistors 48,000 pixel, contrast sensitive edge enhancing, cmos imager-silicon retina. In *Proc. 16th Conf. Advanced Research in VLSI*, pages pp. 225–240, 1995.

[BR96] M. Black and A. Rangarajan. On the unification of line processes, outlier rejection, and robust statistics with applications in early vision. *Int. J. Computer Vision*, 19(1):75–104, 1996.

[CBFAB97] P. Charbonnier, L. Blanc-Féraud, G. Aubert, and M. Barlaud. Deterministic edge-preserving regularization in computed imaging. *IEEE Trans. Image Processing*, 6(2):298–311, 1997.

[DQ88] Vincent Van Dongen and Patrice Quinton. Uniformization of linear recurrence equations: a step towards the automatic synthesis of systolic array. In K. Bromley et al. eds., editor, *International Conference on Systolic Arrays*, pages 473–482. IEEE Computer Society Press, 1988.

[DR94] Alain Darte and Yves Robert. Constructive methods for scheduling uniform loop nests. *IEEE Transactions on Parallel Distributed Systems*, 5.8:814–822, 1994.

[FA92] R. Forchhmeimer and A. Astrom. Mapp2200: a second generation smart optical sensor. In *Proc. SPIE, Vol. 1659, Image Processing and Interchange*, 1992.

[GG84] S. Geman and D. Geman. Stochastic relaxation, Gibbs distributions and the Bayesian restoration of images. *IEEE Trans. Pattern Anal. Machine Intell.*, 6(6):721–741, 1984.

[GM87] S. Geman and D. Mcclure. Statistical methods for tomographic image reconstruction. In *Bull. Isi, Proc. 46th Session Int. Statistical Institute*, volume 52, 1987.

[GR92] D. Geman and G. Reynolds. Constrained restoration and the recovery of discontinuities. *IEEE Trans. Pattern Anal. Machine Intell.*, 14(3):367–383, 1992.

[Gre90] P.J. Green. Bayesian reconstructions from emission tomography data using a modified EM algorithm. *IEEE trans. Med. Imaging*, 9:84–93, 1990.

[Hel94] P.C. Held. Hipars: a tool for automatic conversion of nested loop programs into single assignment programs. Technical Report Dept. Electrical Engineering, Delft University of Technology, 1994.

[HS81] B. Horn and B. Schunck. Determining optical flow. *Artificial Intelligence*, 17:185–203, 1981.

[Hub81] P. Huber. *Robust Statistics*. John Wiley & Sons, 1981.

[HW77] P. Holland and R. Welsch. Robust regression using iteratively reweighted least-squares. *Commun. Statis.-Theor. Meth.*, A6(9):813–827, 1977.

[Lec89] Y. Leclerc. Constructing simple stable descriptions for image partitioning. *Int. J. Computer Vision*, 3:73–102, 1989.

[Mau89] C. Mauras. *Alpha : un langage équationnel pour la conception et la programmation d'architectures parallèles synchrones*. PhD thesis, Université de Rennes 1, IFSIC, December 1989.

[MF86] D.I. Moldovan and J.A.B. Fortes. Partitioning and mapping algorithms into fixed-size systolic arrays. *IEEE Transactions on Computers*, 35(1):1–12, jan 1986.

[MKF94] S. Mendis, S.E. Kemeny, and E.R. Fossum. Cmos active pixel sensor. *IEEE Trans. on Electon. Devices*, 41(3):452–453, march 1994.

[Mol82] D.I. Moldovan. On the analysis and synthesis of vlsi systolic arrays. *IEEE Transactions on Computers*, 31:1121–1126, 1982.

[MR98] E. Memin and T. Risset. Hardware driven considerations for VLSI derivation of discontinuity preserving energy based applications. Technical Report 1220, IRISA, Rennes, France, 1998. available at `http://www.irisa.fr/EXTERNE/bibli/pi/1220/1220.html`.

[PTK87] T. Poggio, V. Torre, and C. Koch. Computational vision and regularization theory. In M.A. Fischler and O. Firschein, editors, *Readings In Computer Vision : Issues, Problems, Principles and Paradigms*, pages 638–643. Morgan Kaufmann, 1987.

[QR89] Patrice Quinton and Yves Robert. *Systolic Algorithms and Architectures*. Prentice Hall and Masson, 1989.

[RD96] L. Robert and R. Deriche. Dense depth map reconstruction: a minimization and regularization approach which preserves discontinuities. In B. Buxton and R. Cipolla, editors, *Proc. Europ. Conf. Computer Vision*, number 1064 in LNCS, pages 439–451. Springer-Verlag, April 1996.

[tHR94] B. M. ter Haar Romeny, editor. *Geometry-driven diffusion in computer vision*. Kluwer academic, 1994.

[TM86] J. Tanner and C. Mead. An integrated analog optical motion sensor. In S.Y. Kung, R. Owen, and J.G. Nash, editors, *VLSI signal processing II*. IEEE Press, 1986.

[Wil94] D. Wilde. The alpha language. Technical Report 827, IRISA, Rennes, France, Dec 1994.