Full alternate jacobi Minimization and vlsi derivation for motion estimation

Etienne Memin, tanguy Risset
Valoria (Vannes), Irisa (Rennes)
France

Outline

- Introduction/motivation
- Energy minimization method:
  - IRLS
  - FAJM
  - Convergence: test and proof
- Hardware derivation:
  - methodology
  - Synthesis results

Introduction/Motivation

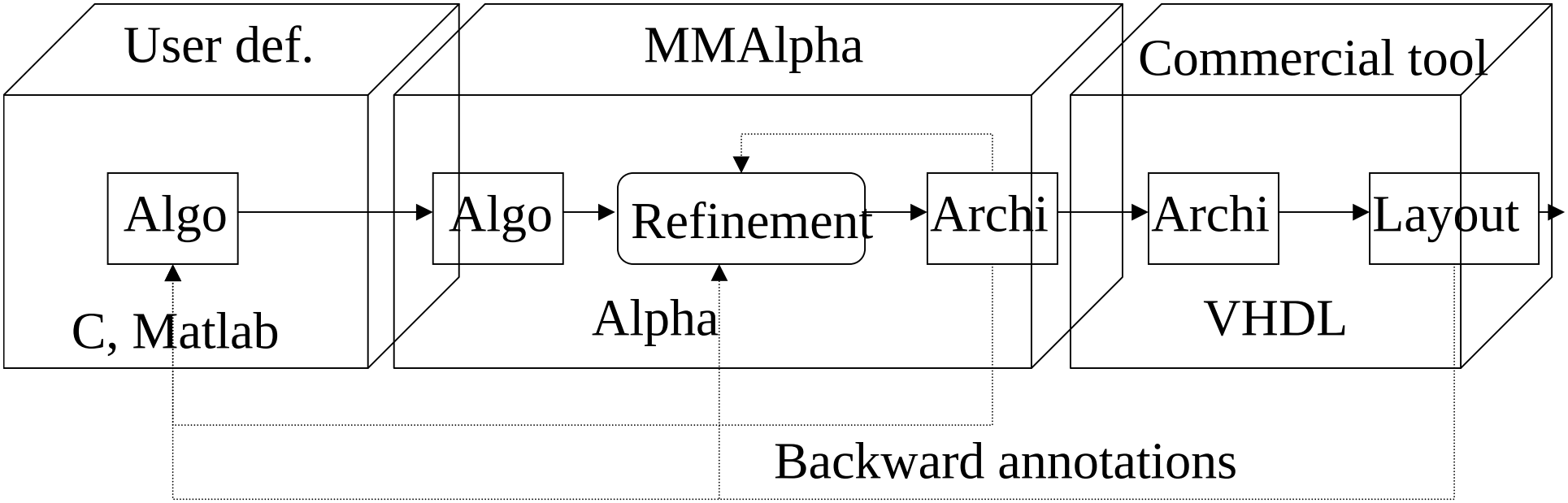
- Many problems use minimization of energy
- Algorithms are:
  - computationally intensive,
  - regular, parallel,
  - in constant development,
- Real-time requires hardware (VLSI or FPGA) and fast design (time to market)

Introduction/Motivation (2)

- Hardware design ≠ algorithm design
  - ≠ constraints and cost functions
  - ≠ programming languages
  - ≠ scientific background
- A good design should be:
  - optimal in all sense
  - easy to modify/update
  - rapidly prototyped

Our solution: synthesis with MMAlpha

Algorithm design High level synthesis Logic synthesis



Discontinuity preserving energy minimization

- Minimization: $\hat{w} = Argmin(H(w))$ where

$$H(w) = \underbrace{\frac{1}{2}\left\| f - \kappa w \right\|^2}_{\text{data term}} + \underbrace{\frac{\alpha}{2}\sum_{(s,r)\in C}\rho\left(\left\| w_s - w_r \right\|^2\right)}_{\text{regularization term}}$$

- f: observed data, α κ: constants, C: set of neighboring points, ρ : robust estimator

Introducing adaptive weights

- Under certain conditions, we can have:

$$H(w) = \min_{\beta}(H^*(w,\beta))$$

- Where $H^*(w,\beta)$ is:

$$\frac{1}{2}\left\| f - \kappa w \right\|^2 + \frac{\alpha}{2}\sum_{(s,r)\in C}\tau\beta_{s,r}\left\| w_s - w_r \right\|^2 + \psi\left(\beta_{s,r}\right)$$

Iterative Reweighted Least Square

1- Fix the weights (β^j) and compute:

$$w^{j+1} = \underset{w}{Argmin}(H^*(w, \beta^j))$$

by iterative method (Jacobi or Gauss Siedel).

2- Fix w^j and compute explicitly β^{j+1}

$$\beta^{j+1} = \underset{\beta}{Argmin}(H^*(w^{J+1}, \beta))$$

Example with Jacobi

- In matrix notation we have:

$$H^*(w,\beta)=\frac{1}{2}w^T A_\beta w - b^T w + c_\beta$$

- With Jacobi step 1 is:

$$w_{k+1}^j = w_k^j + p_k^j \qquad \text{with } p_k^j = \mathrm{D}_\beta^{-1}\left(b - A_\beta w_k^j\right)$$

$$w^{j+1} = \lim_{k\to\infty} w_k^j$$

Part I: FAJM Method

- Hardware derivation remarks:
  - Jacobi is well adapted to massive parallelism.
  - Hardware for updating the weights is necessary.
  - Complicated control procedure for testing convergence at each step.

$\Longrightarrow$Modification of IRLS into FAJM

Full Alternate Jacobi Minimization

- principle: update the weights at each Jacobi steps

$$\begin{cases} w^{k+1} = w^k + p^k \qquad \text{with } p^k = \mathrm{D}_{\beta_k}^{-1}\left(b - A_{\beta_k} w^k\right) \\ \beta^{k+1} = \underset{\beta}{Argmin}(H^*(w^{k+1}, \beta)) \end{cases}$$

- New method $\Longrightarrow$need of proof and evaluation.

FAJM: convergence proof

- Currently we have proved that:
  - FAJM converges
  - If FAJM converges towards (w_0,β_0), then this point is a stationary point for H^*:

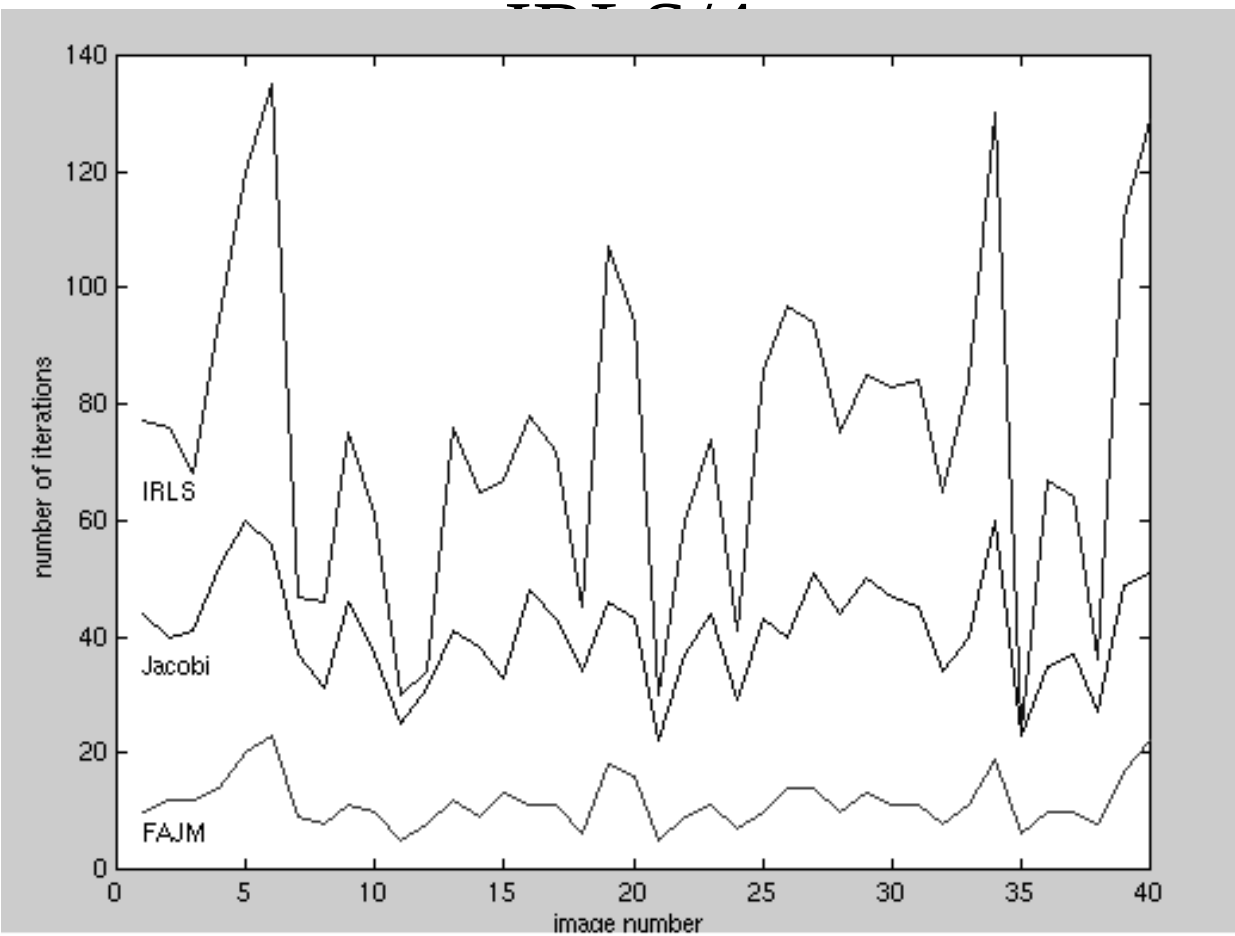
$$\nabla H^*(w_0,B_0)=0;$$

FAJM: convergence experiments

- Experiments conducted on image restoration and optical flow estimation.
- Benchmarks of 40 images (64x64), with two robust estimators
- Comparison of the simple Jacobi method (JM) and the two Jacobi based alternate minimization (IRLS and FAJM).

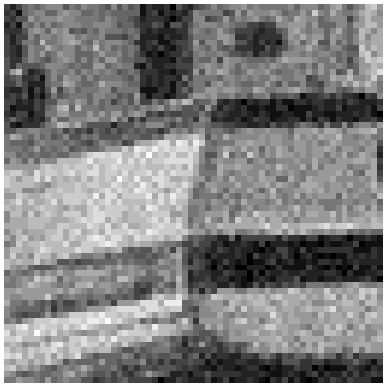
FAJM: Converge speed

- Number of iteration: FAJM⇔JM /2 ⇔

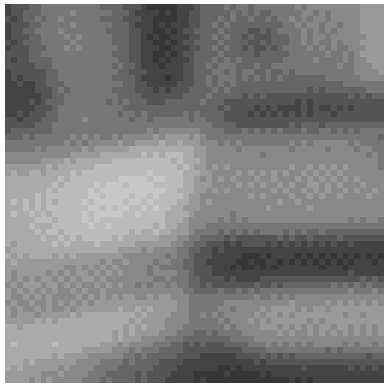


FAJM: quality of result

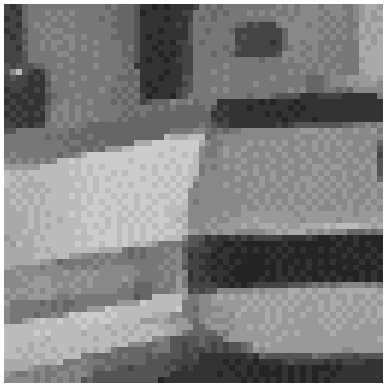
- Best minimization: FAJM$\lesssim$IRLS$<<<$JM
- Example on image restoration:



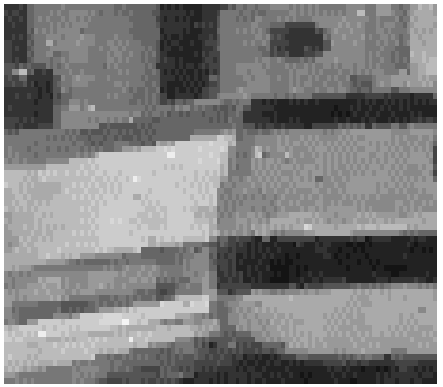
Image



JM



IRLS



FAJM

FAJM: evaluation

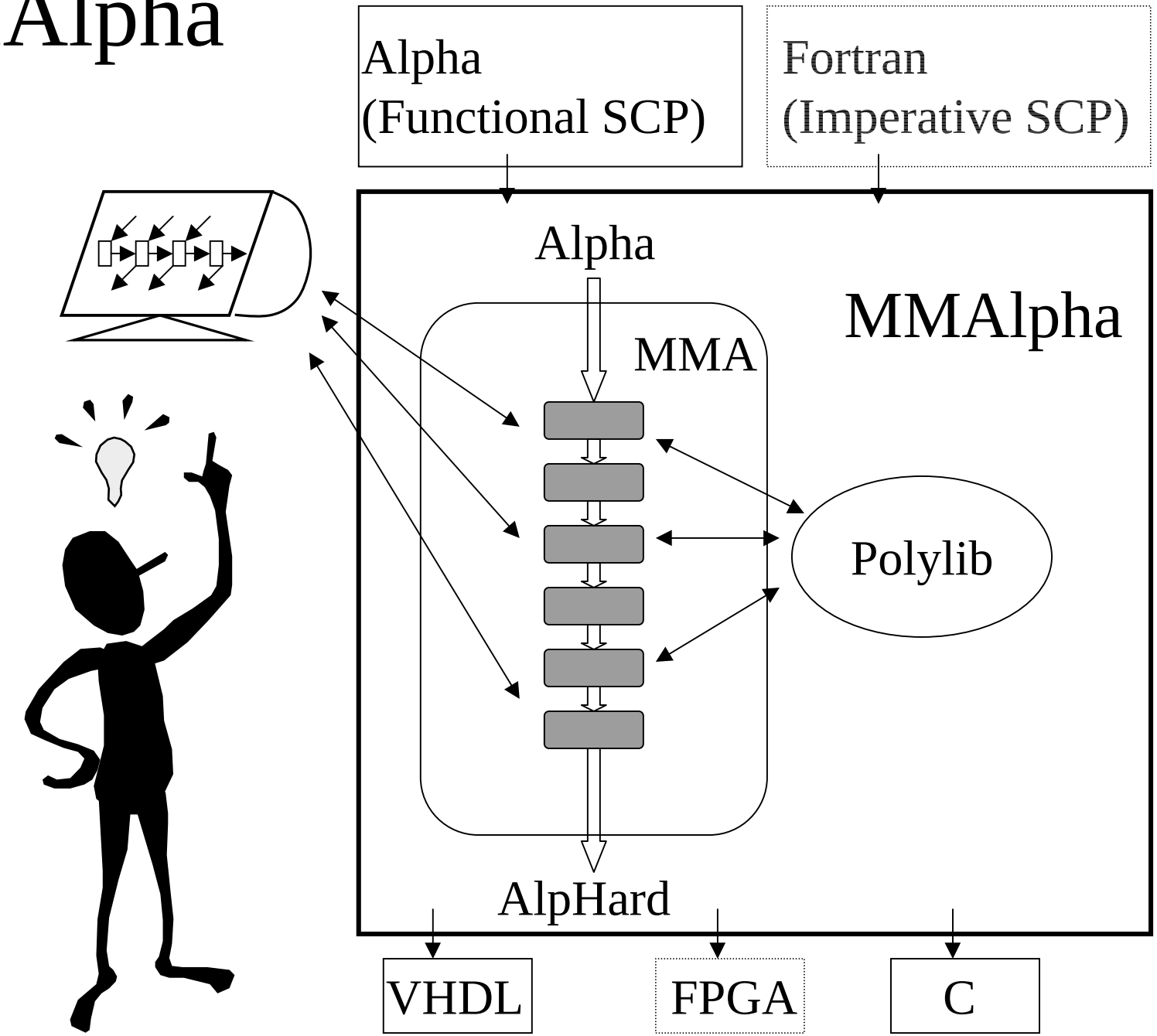
- The FAJM is :
  - 4 times faster than existing IRLS
  - At least as efficient as IRLS
  - simpler to implement

$\Longrightarrow$Suitable for hardware and software implementations

Part II: Hardware derivation

- Use of MMAlpha: (http://www.irisa.fr/api/ALPHA)
  1) Select a high level specification.
  2) Translate it into Alpha.
  3) Transform the Alpha spec. Into Hardware spec. (AlpHard)
  4) Translate it into VHDL
  5) Synthesize VHDL with commercial tools

MMAlpha

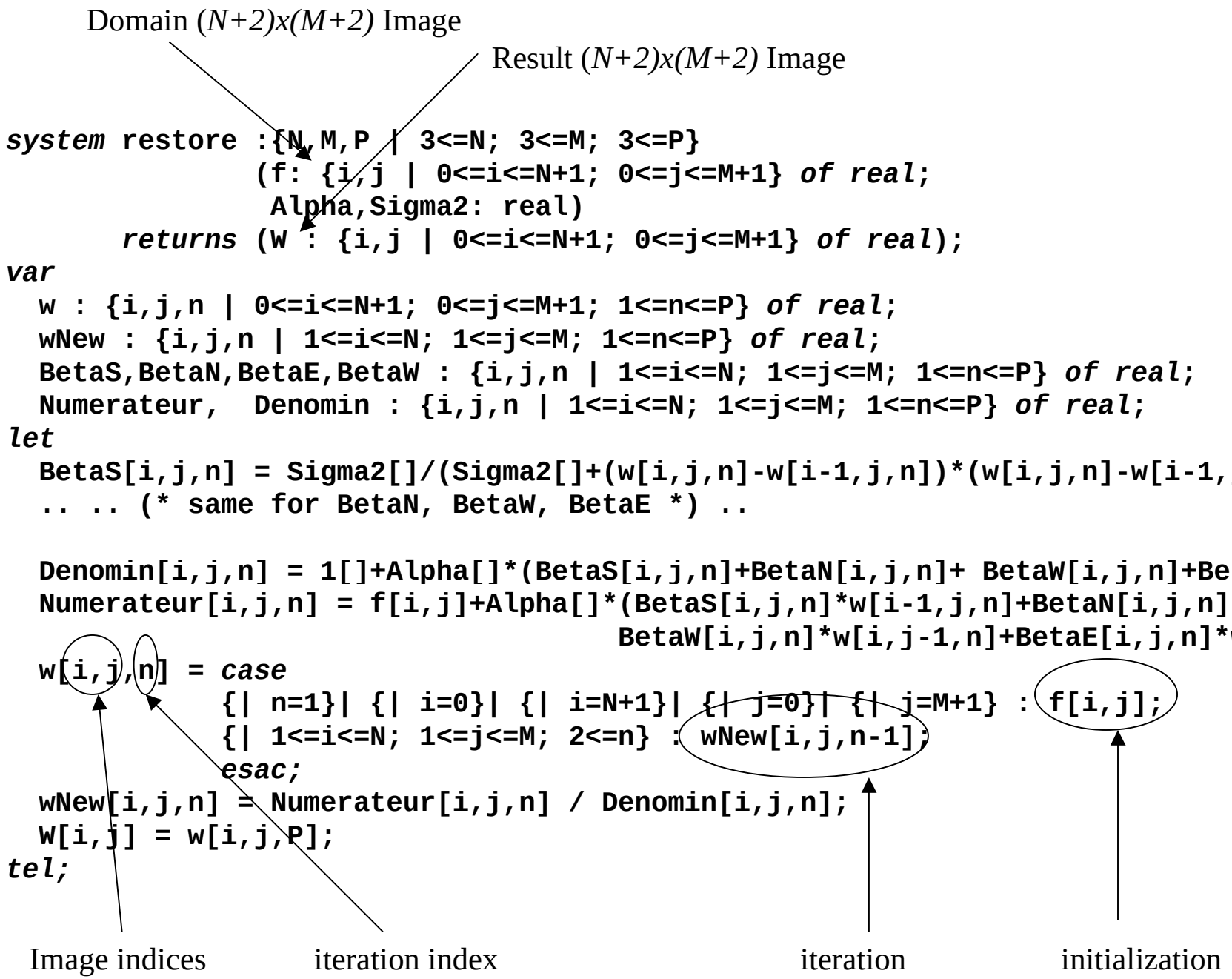

Alpha
(Functional SCP)
Fortran
(Imperative SCP)
Alpha
MMAlpha
MMA
Polylib
AlpHard
VHDL
FPGA
C

Specification: image restoration

$$\forall (i,\ j, n) \in \left\{ i,\ j, n \middle| 1 \le i,\ j \le m; 1 \le n \le P \right\}$$

$$\kappa \in \left\{ \underbrace{(-1,0)}_{N}, \underbrace{(0,-1)}_{W}, \underbrace{(1,0)}_{S}, \underbrace{(0,1)}_{E} \right\}$$

$$\begin{cases} w(i,\ j, n+1) = \dfrac{f(i,\ j) + \alpha \sum\limits_{\kappa} \beta_{\kappa}(i,\ j, n) w[(i,\ j) + \kappa, n]}{1 + \alpha \sum\limits_{\kappa} \beta_{\kappa}(i,\ j, n)} \\ \beta(i,\ j, n+1) = \dfrac{\sigma^2 \rho'(w(i,\ j, n+1) - w[(i,\ j) + \kappa, n+1])}{2(w(i,\ j, n+1) - w[(i,\ j) + \kappa, n+1])} \end{cases}$$


Domain (N+2)x(M+2) Image
Result (N+2)x(M+2) Image
system restore :{N,M,P | 3<=N; 3<=M; 3<=P}
(f: {i,j | 0<=i<=N+1; 0<=j<=M+1} of real;
Alpha,Sigma2: real)
returns (W : {i,j | 0<=i<=N+1; 0<=j<=M+1} of real);
var
w : {i,j,n | 0<=i<=N+1; 0<=j<=M+1; 1<=n<=P} of real;
wNew : {i,j,n | 1<=i<=N; 1<=j<=M; 1<=n<=P} of real;
BetaS,BetaN,BetaE,BetaW : {i,j,n | 1<=i<=N; 1<=j<=M; 1<=n<=P} of real;
Numerateur, Denomin : {i,j,n | 1<=i<=N; 1<=j<=M; 1<=n<=P} of real;
let
BetaS[i,j,n] = Sigma2[]/(Sigma2[]+(w[i,j,n]-w[i-1,j,n])*(w[i,j,n]-w[i-1,j,n]));
.. .. (* same for BetaN, BetaW, BetaE *) ..
Denomin[i,j,n] = 1[]+Alpha[]*(BetaS[i,j,n]+BetaN[i,j,n]+ BetaW[i,j,n]+BetaE[i,j,n])
Numerateur[i,j,n] = f[i,j]+Alpha[]*(BetaS[i,j,n]*w[i-1,j,n]+BetaN[i,j,n]*w[i+1,j,n]
BetaW[i,j,n]*w[i,j-1,n]+BetaE[i,j,n]*w[i,j+1,n])
w[i,j,n] = case
{| n=1}| {| i=0}| {| i=N+1}| {| j=0}| {| j=M+1} : f[i,j];
{| 1<=i<=N; 1<=j<=M; 2<=n} : wNew[i,j,n-1];
esac;
wNew[i,j,n] = Numerateur[i,j,n] / Denomin[i,j,n];
W[i,j] = w[i,j,P];
tel;
Image indices
iteration index
iteration
initialization

Hardware derivation methodology

1) Uniformization (for local communications)
   – alignment, pipeline, routing.
2) scheduling/allocation (parallelization)
   – get a virtual array
3) I/O routing (to boundary processors)
4) Control signal generation
5) Formatting Alphard,
   – simulate/evaluate design.

Uniformization

- Replace :

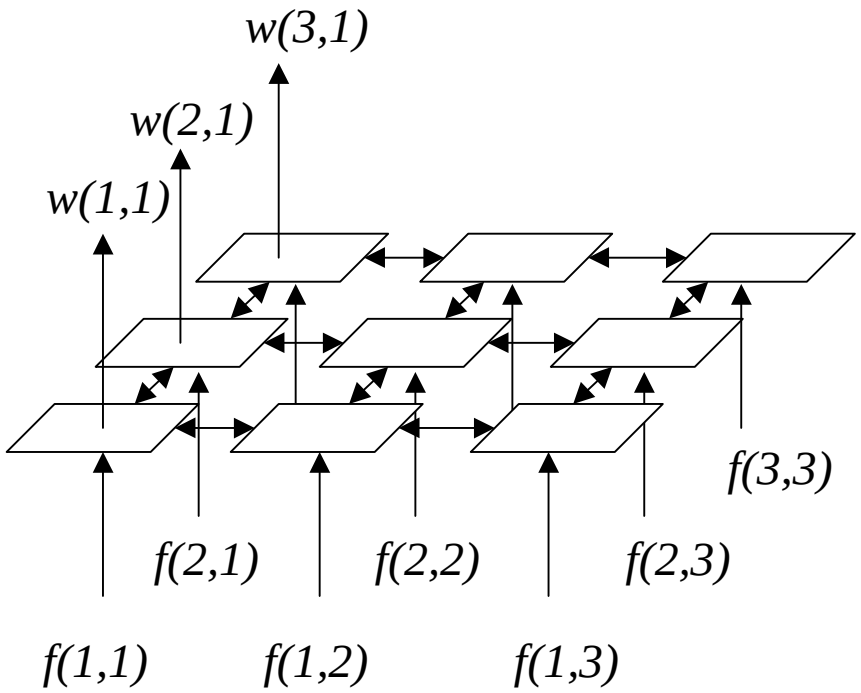
```
Numerateur[i,j,n] = f[i,j]
```

- By :

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

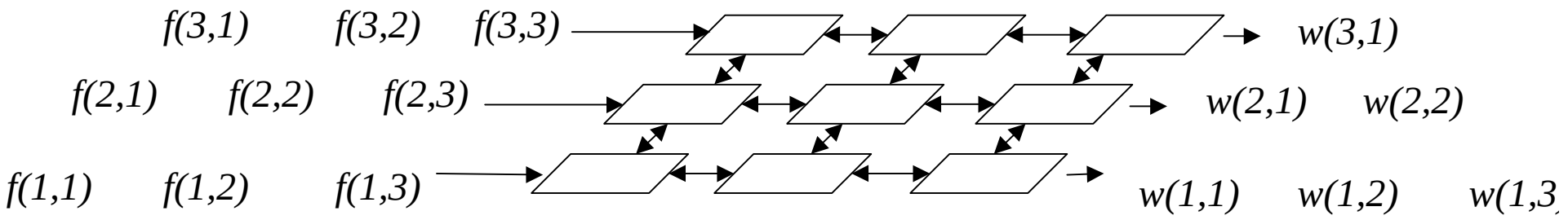
Scheduling/Allocation

- Index *(i,j,n)* executed at step *n* on processor *(i,j)*.



I/O routing

- Pipelining input and outputs to I/O processors



Control signal generation

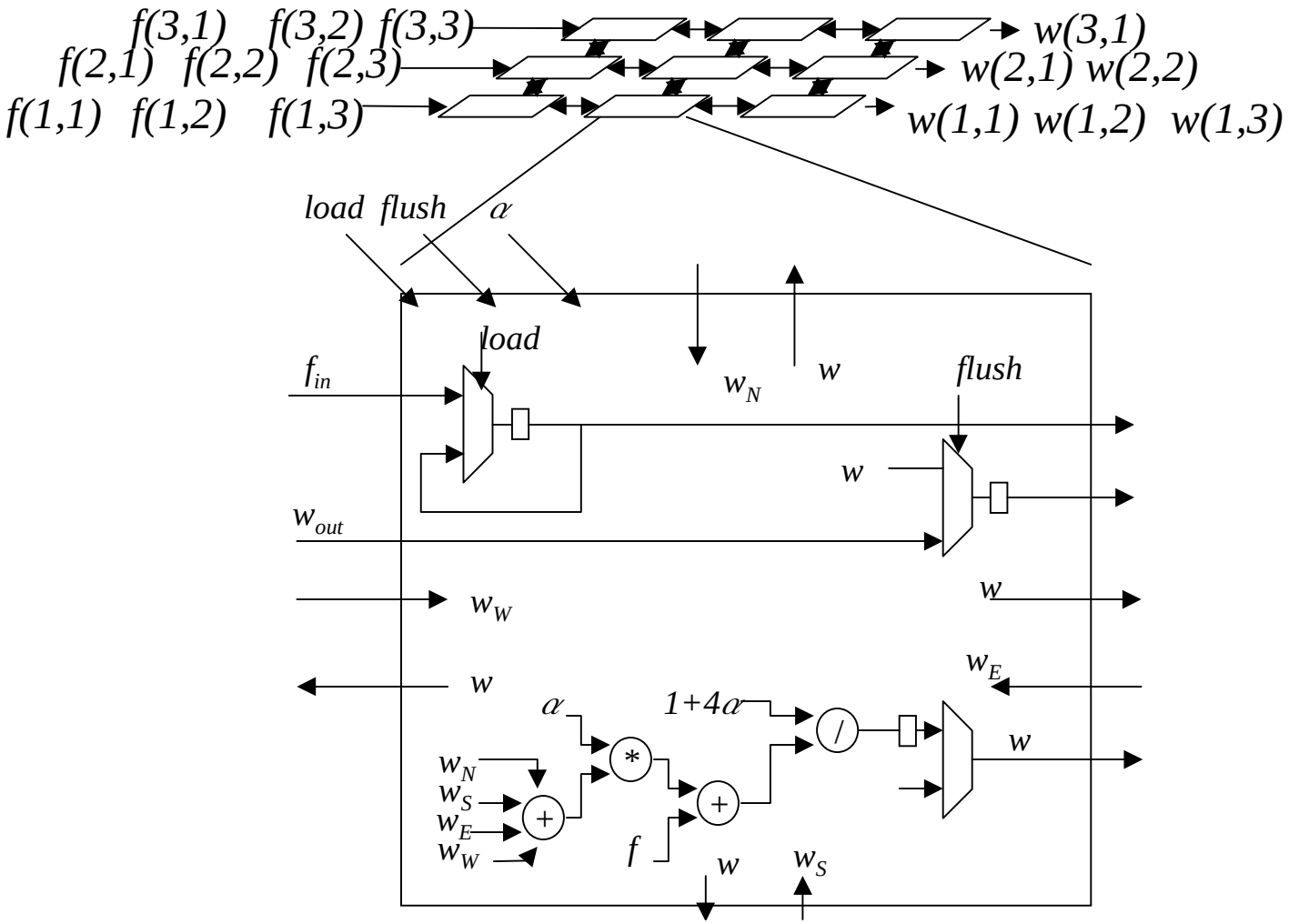
- The behavior of the virtual processors is described by:

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

- We must replace the software control by hardware control (i.e. control signal initialized by the host):

```
Pipef[t,p1,p2] = If ctl_Pipef[t]
                    then f[p2,p1]
                    else Pipef[t-1,p1,p2];
```

Alphard architecture (JM)



Hardware synthesis results

- Experiments on:
  - Image restoration and motion estimation,
  - Jacobi and FAJM.
- Bit level simulation $\Longrightarrow$bit width for signals.
- Synthesis with Compass tool
  - λ=0.4 μ,
  - Automatic synthesis/ placing / routing.

Hardware synthesis for restoration

	Size of 1 cell	Clock cycle	Convergence time
Jacobi (8 bits)	0.12 mm²	25 ns	9.5 µs
Jacobi (14 bits)	1.4 mm²	50 ns	11 µs
FAJM (8 bits)	1.16 mm²	50 ns	11 µs
FAJM (14 bits)	21.5 mm²	100 ns	22 µs

Hardware synthesis for estimation

	Size of 1 cell	Clock cycle	Convergence time
Jacobi (8 bits)	0.4 mm²	37 ns	9.2 µs
Jacobi (14 bits)	1.4 mm²	56 ns	14 µs
FAJM (8 bits)	9.52 mm²	89 ns	22 µs
FAJM (14 bits)	24.8 mm²	117 ns	29 µs

Hardware synthesis results

- Hardware for Jacobi motion estimation of 64x64 image:
  - 7.5 cm x 7.5 cm,
  - 10^5 image/s.
- Hardware for FAJM image restoration of 10x10 image:
  - 5 cm x 5 cm,
  - 5.10^4 image/s.
- Too big, too fast $\Longrightarrow$ Partitioning.

Conclusion

- Results:
  - Semi automatic high level synthesis for edge preserving energy minimization problems.
  - New efficient method for the resolution.
- Future:
  - Not integrated enough for current hardware implementation (wait 2-3 years)
  - size and I/O Bottleneck $\Longrightarrow$partitioning.
  - Ideal approach for prototyping on FPGA