Un Cluster pour la Vision Temps Réel

Architecture, Outils et Applications

Joël Falcou

sous la direction de Jocelyn SÉROT, Thierry CHATEAU et Jean-Thierry LAPRESTÉ

LASMEA - UMR CNRS 6602
Équipe GRAVIR
63170 Aubière

1er Décembre 2006

Contexte des travaux

La vision artificielle

Interpréter automatiquement le contenu d'une ou plusieurs images afin d'en extraire des informations de haut-niveau.

Contexte des travaux

La vision artificielle

Interpréter automatiquement le contenu d'une ou plusieurs images afin d'en extraire des informations de haut-niveau.

Champs d'applications

- Robotique mobile.
- Réalité virtuelle ou augmentée.
- Systèmes de surveillance.

Contexte des travaux

La vision artificielle

Interpréter automatiquement le contenu d'une ou plusieurs images afin d'en extraire des informations de haut-niveau.

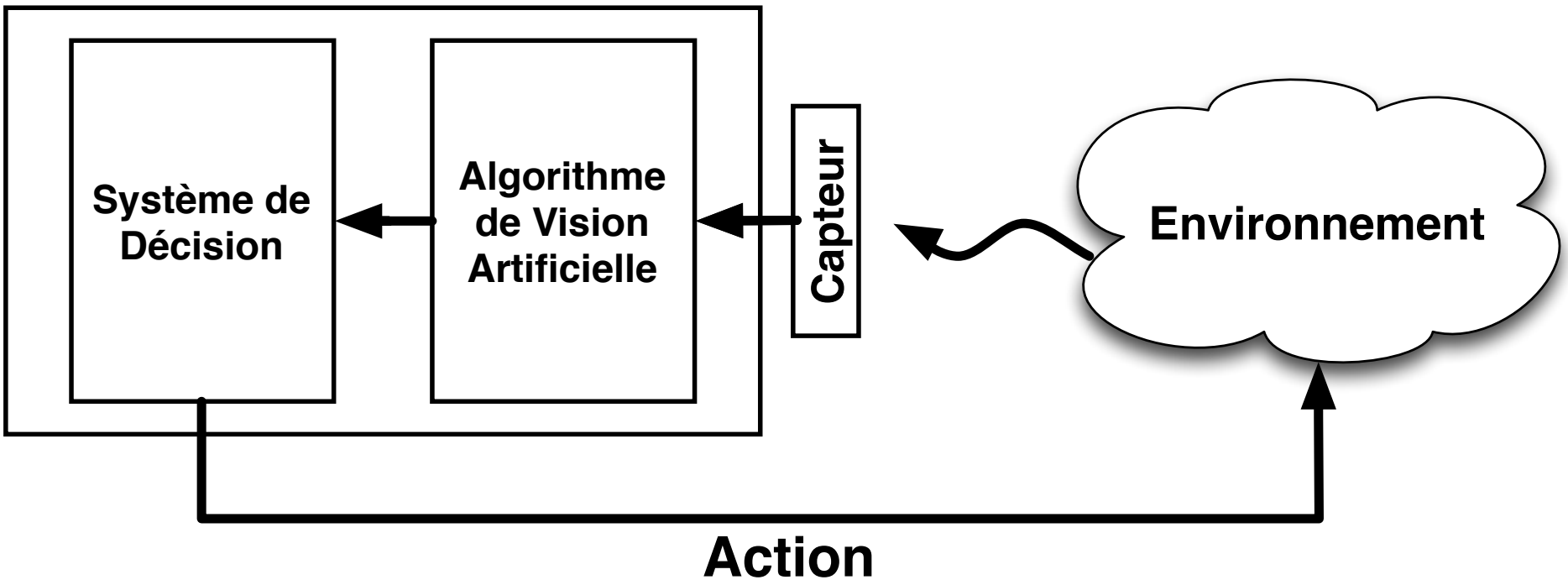
Champs d'applications

- Robotique mobile.
- Réalité virtuelle ou augmentée.
- Systèmes de surveillance.

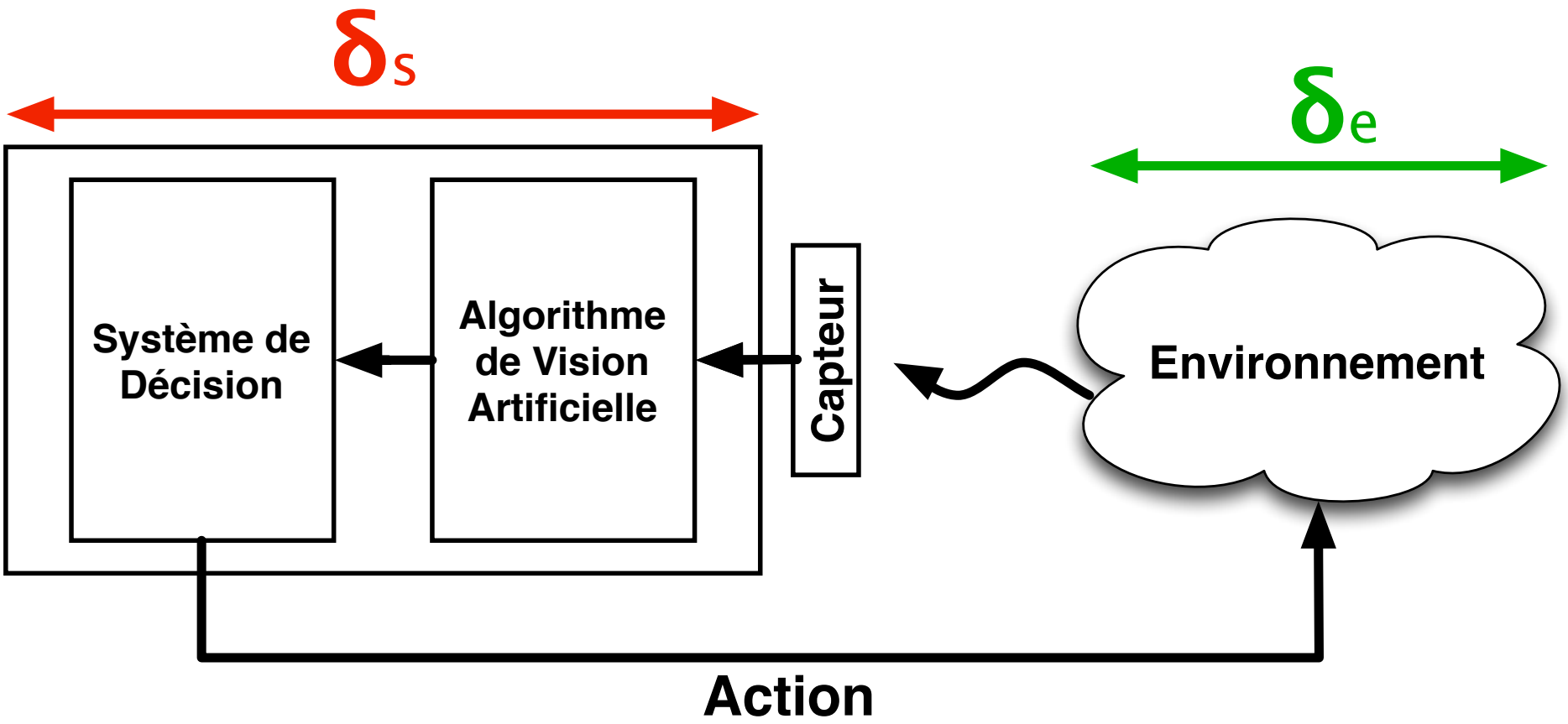
Problématique

Les algorithmes de vision artificielle sont rarement utilisés seuls.

La vision artificielle temps réel



La vision artificielle temps réel



Objectifs

Applications visées

- Consommatrices de temps de calcul.
- Contraintes du temps réel vidéo.

Objectifs

Applications visées

- Consommatrices de temps de calcul.
- Contraintes du temps réel vidéo.

Constat

L'utilisation «basique» des ordinateurs standards ne suffit pas pour satisfaire ces contraintes temporelles.

Objectifs

Applications visées

- Consommatrices de temps de calcul.
- Contraintes du temps réel vidéo.

Constat

L'utilisation «basique» des ordinateurs standards ne suffit pas pour satisfaire ces contraintes temporelles.

Déroulement des travaux

- Développer une architecture adaptée à la Vision Temps Réel.
- Définir des outils de programmation adaptés.
- Démontrer la pertinence des solutions proposées.

Plan de l'exposé

Plan de l'exposé

Plan de l'exposé

Plan de l'exposé

Plan de l'exposé

Les architectures pour la vision

Les architectures spécifiques

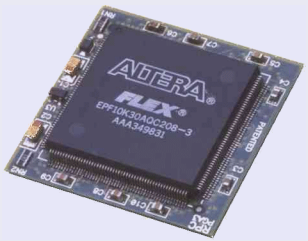
- Circuits intégrés dédiés à une fonctionnalité (ASIC).
- Circuits intégrés reprogrammables (FPGA).
- Processeurs de cartes graphiques (GPU).

Les architectures pour la vision

Les architectures spécifiques

- Circuits intégrés dédiés à une fonctionnalité (ASIC).
- Circuits intégrés reprogrammables (FPGA).
- Processeurs de cartes graphiques (GPU).

Spécificités



- Performances élevées.
- Outils de développement complexes.
- Solutions difficilement réutilisables.

Les architectures pour la vision

Grappe de calcul (*Cluster*)

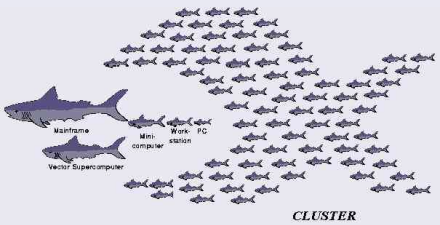
Machine parallèle composée d'un ensemble de calculateurs individuels interconnectés travaillant comme une unique ressource de calcul.

Les architectures pour la vision

Grappe de calcul (*Cluster*)

Machine parallèle composée d'un ensemble de calculateurs individuels interconnectés travaillant comme une unique ressource de calcul.

Spécificités



- Plate-forme polyvalente.
- Bon rapport performance/coût.
- Outils et modèles standards.

Quelques *clusters* pour la vision

État de l'art

- The 3D Room [KANADE 96].
- ViROOM [SVOBODA 02].
- GrImage [BOYER 05].

Quelques *clusters* pour la vision

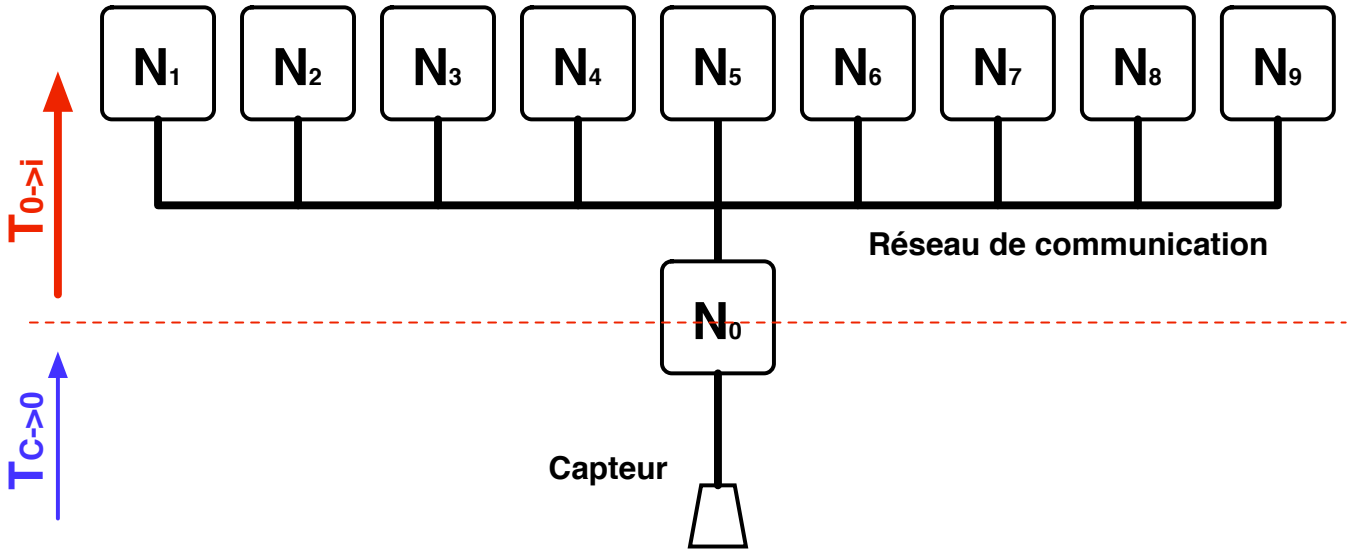
État de l'art

- The 3D Room [KANADE 96].
- ViROOM [SVOBODA 02].
- GrImage [BOYER 05].

Problèmes

- Architectures dédiées au problème traité.
- Programmation «basique» des nœuds de calcul.

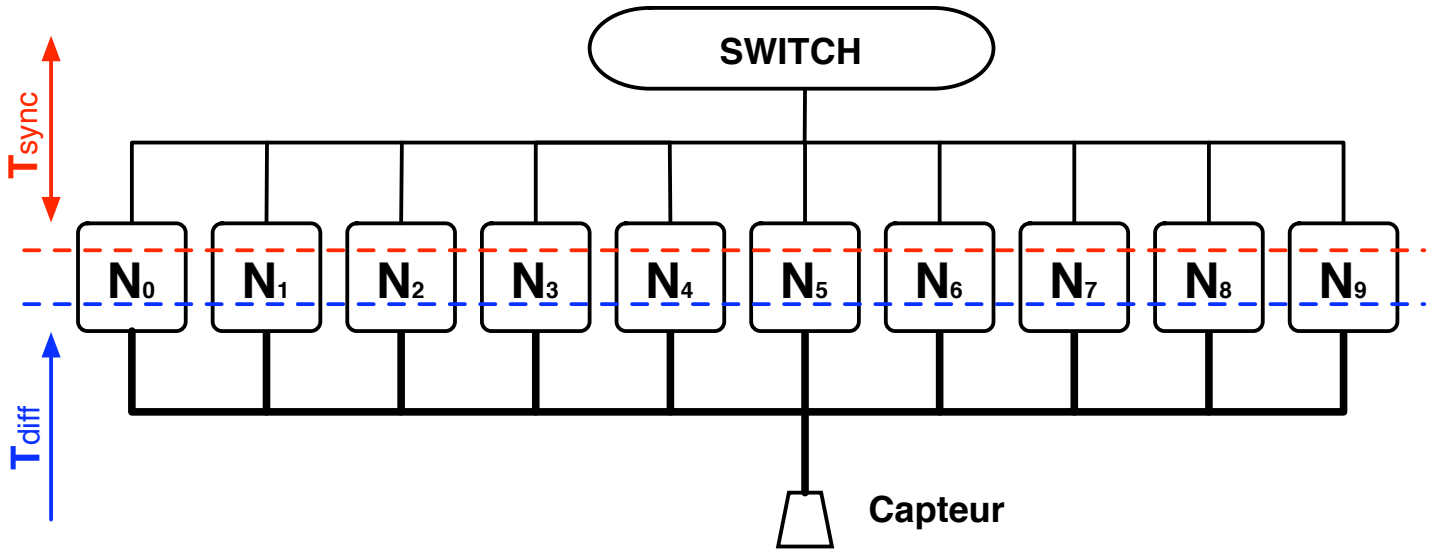
Architecture classique d'un *cluster*



Aspects temporels

- Temps d'acquisition sur N_0 ($T_{C\rightarrow 0}$)
- Temps de transfert de N_0 vers N_i ($T_{0\rightarrow i}$)
- Temps de mise à disposition : $T_{C\rightarrow 0} + P \times T_{0\rightarrow i}$

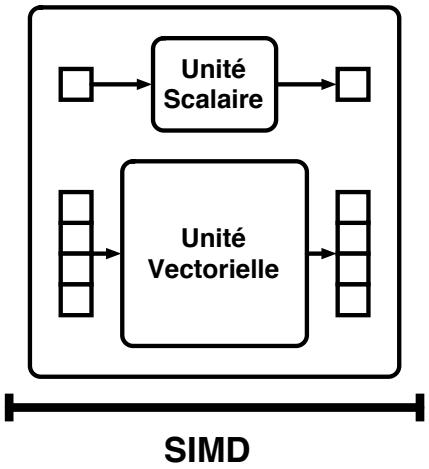
Architecture à double bus



Aspects temporels

- Temps de synchronisation (T_{sync})
- Temps de diffusion sur le bus video (T_{diff})
- Temps de mise à disposition : $T_{sync} + T_{diff}$

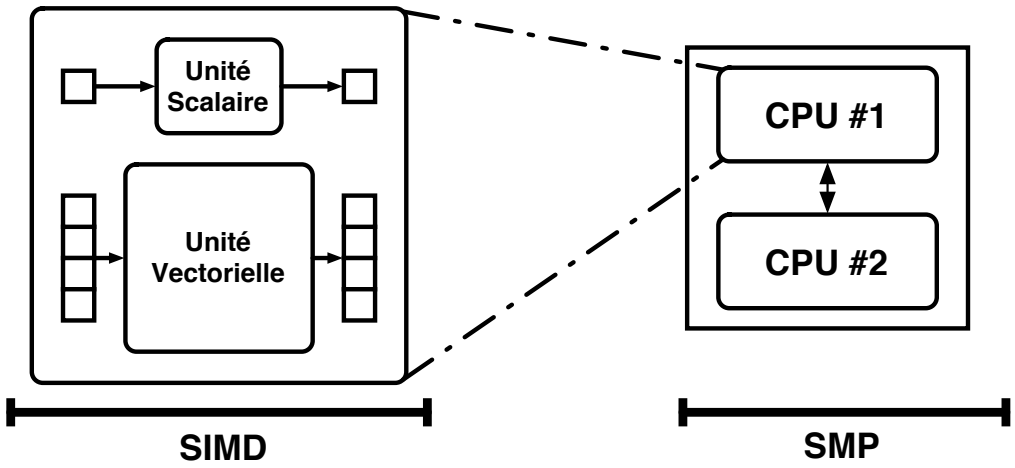
Exploitation des nœuds de calcul



Les extensions multimédia SIMD (*Single Instruction Multiple Data*)

- Unité de traitement SIMD interne au processeur.
- Accélération maximale théorique dépendante du type de donnée.
- En pratique $V \approx 4 - 16$.

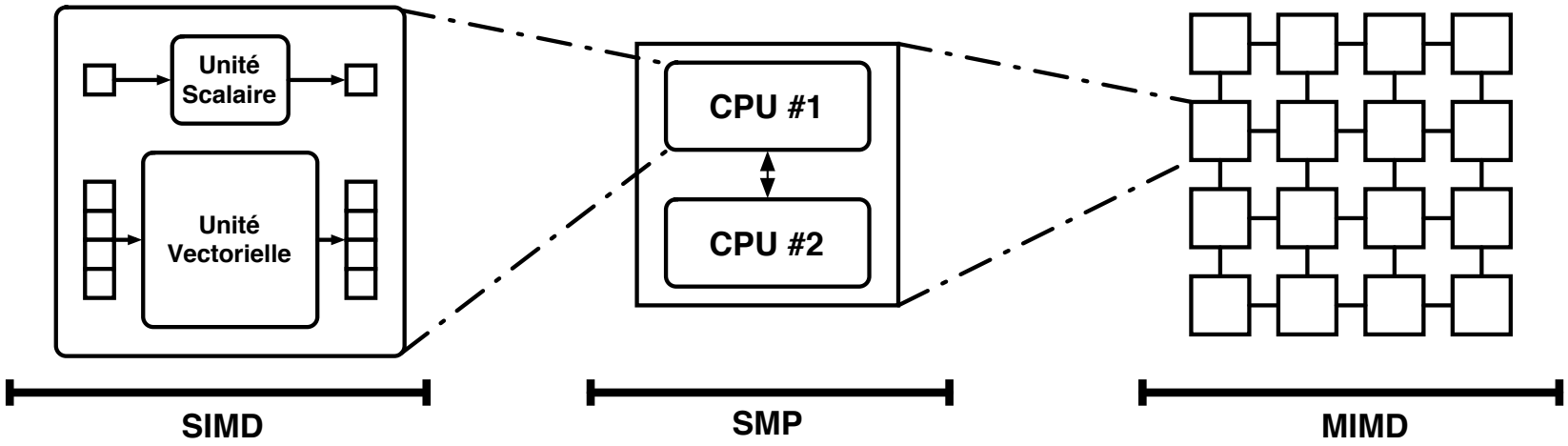
Exploitation des nœuds de calcul



Les architectures SMP (*Symetric Multi-Processor*)

- P processeurs par nœud de calcul
- Accélération maximale théorique $\approx P \times V$
- Machines bi-processeur $\approx 8 - 32$

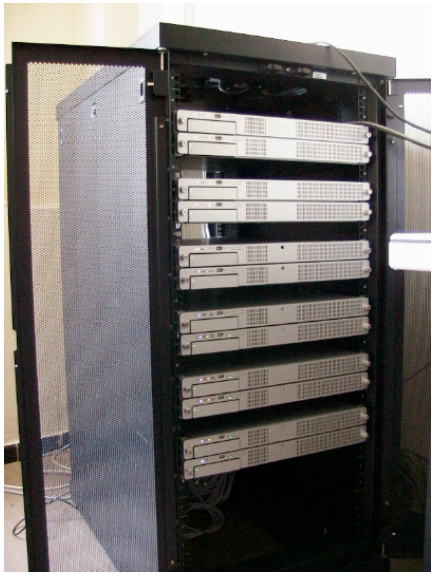
Exploitation des nœuds de calcul



Les architectures MIMD (*Multiple Instructions Multiple Data*)

- N machines à P processeurs + extension SIMD
- Accélération maximale théorique $\approx N \times P \times V$
- Cluster de 16 machines $\approx 128 - 512$

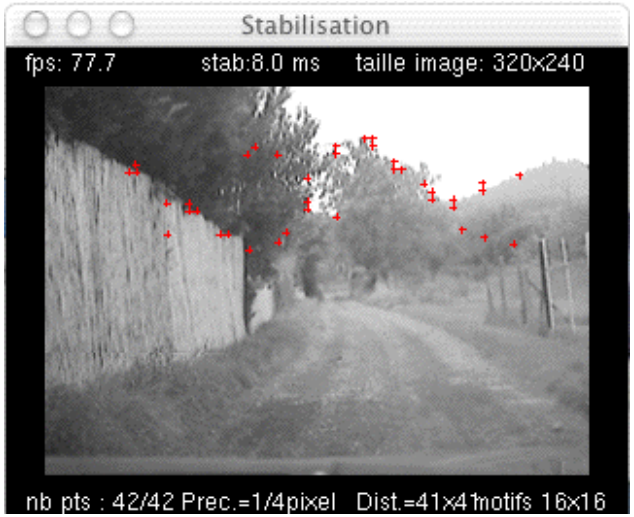
BABYLON



Une instance de *cluster* double bus [*ParCo/GRETSI 2005*]

- 14 PowerPC G5 (2x2GHz + AltiVec).
- Réseau Ethernet Gigabit.
- Bus vidéo Firewire IEEE1394.

Application de stabilisation d'image



Temps réel vidéo

- Pour une programmation «basique» jusqu'à 320x240 pixels.
- Sur BABYLON, jusqu'à 5120x3840 pixels
- Accélération de 35 à 50

```
//-----------------------------------------------
//detection of most easy points to track
//-----------------------------------------------

local_detected_pts_number=0;  //reset detection

top1 = mtime();
if(!first_iteration)
{
  Detection(INT_img_pred,DETBORDER_left,DETBORDER_up,DETBORDER_right,DETBORDER_bottom,
            local_Detected_point_x,local_Detected_point_y,&local_detected_pts_number,local_wanted_pts_number);
}
top2 = mtime();
timer[3] = top2-top1;

//-----------------------------------------------
//tracking of selected points
//-----------------------------------------------
top1 = mtime();

TrackingPoints(local_Detected_point_x,local_Detected_point_y,local_Tracked_point_x,local_Tracked_point_y,local_detected_pts_number,
img_pred.data,img_curr.data,
LR2_img_pred.data,LR2_img_curr.data,
LR4_img_pred.data,LR4_img_curr.data,
TX,TY);

SwapImages();

MPI_Barrier(MPI_COMM_WORLD);

MPI_Gather( local_Detected_point_x, local_wanted_pts_number, MPI_DOUBLE, Detected_point_x, local_wanted_pts_number, MPI_DOUBLE,MASTER,
MPI_Gather( local_Detected_point_y, local_wanted_pts_number, MPI_DOUBLE, Detected_point_y, local_wanted_pts_number, MPI_DOUBLE,MASTER,
MPI_Gather( local_Tracked_point_x, local_wanted_pts_number, MPI_DOUBLE, Tracked_point_x, local_wanted_pts_number, MPI_DOUBLE,MASTER, MP
MPI_Gather( local_Tracked_point_y, local_wanted_pts_number, MPI_DOUBLE, Tracked_point_y, local_wanted_pts_number, MPI_DOUBLE,MASTER, MP

MPI_Reduce(&local_detected_pts_number, detected_pts_number,1, MPI_INT, MPI_SUM, MASTER, MPI_COMM_WORLD);

MPI_Allgather( local_Detected_point_x, local_wanted_pts_number, MPI_DOUBLE, Detected_point_x, local_wanted_pts_number, MPI_DOUBLE, MPI_
MPI_Allgather( local_Detected_point_y, local_wanted_pts_number, MPI_DOUBLE, Detected_point_y, local_wanted_pts_number, MPI_DOUBLE, MPI_
MPI_Allgather( local_Tracked_point_x, local_wanted_pts_number, MPI_DOUBLE, Tracked_point_x, local_wanted_pts_number, MPI_DOUBLE, MPI_CO
MPI_Allgather( local_Tracked_point_y, local_wanted_pts_number, MPI_DOUBLE, Tracked_point_y, local_wanted_pts_number, MPI_DOUBLE, MPI_CO

MPI_Allreduce(&local_detected_pts_number, detected_pts_number,1, MPI_INT, MPI_SUM, MPI_COMM_WORLD);
MPI_Allreduce(timer, timer, 10, MPI_FLOAT, MPI_MAX, MPI_COMM_WORLD);


//-----------------------------------------------------
// movement matrix extraction (median square value)
//-----------------------------------------------------

top1 = mtime();
```

Sommaire

Programmer BABYLON

Trois niveaux de parallélisme

- Parallélisme inter-nœuds (MIMD).
- Parallélisme inter-processeurs (SMP).
- Parallélisme intra-processeur (SIMD).

Programmer BABYLON

Trois niveaux de parallélisme

- Parallélisme inter-nœuds (MIMD).
- Parallélisme inter-processeurs (SMP).
- Parallélisme intra-processeur (SIMD).

Des difficultés variées

- Utilisation efficace de chaque type de parallélisme.
- Interopérabilité SIMD/SMP et SMP/MIMD.

Programmer BABYLON

Trois niveaux de parallélisme

- Parallélisme inter-nœuds (MIMD).
- Parallélisme inter-processeurs (SMP).
- Parallélisme intra-processeur (SIMD).

Des difficultés variées

- Utilisation efficace de chaque type de parallélisme.
- Interopérabilité SIMD/SMP et SMP/MIMD.

Outils disponibles

- Expressivité/Efficacité $\Rightarrow$ Choix du modèle de programmation.
- Accessibilité $\Rightarrow$ Choix du langage cible (C++).

Programmation du niveau MIMD

Outils bas-niveau

- Modèle de programmation par passage de message.
- PVM (*Parallel Virtual Machine*)
- MPI (*Message Passing Interface*)

Programmation du niveau MIMD

Outils bas-niveau

- Modèle de programmation par passage de message.
- PVM (*Parallel Virtual Machine*)
- MPI (*Message Passing Interface*)

Modèles de programmation de plus haut niveau

- Squelettes algorithmiques.
- *Design Patterns* parallèles.

Programmation du niveau MIMD

Outils bas-niveau

- Modèle de programmation par passage de message.
- PVM (*Parallel Virtual Machine*)
- MPI (*Message Passing Interface*)

Modèles de programmation de plus haut niveau

- Squelettes algorithmiques.
- *Design Patterns* parallèles.

Solution adoptée

Squelettes algorithmiques (MPI)

Les squelettes algorithmiques parallèles

Principes

- Eléments récurrents de la programmation MIMD.
- Créer une application parallèle = combiner des squelettes.

Les squelettes algorithmiques parallèles

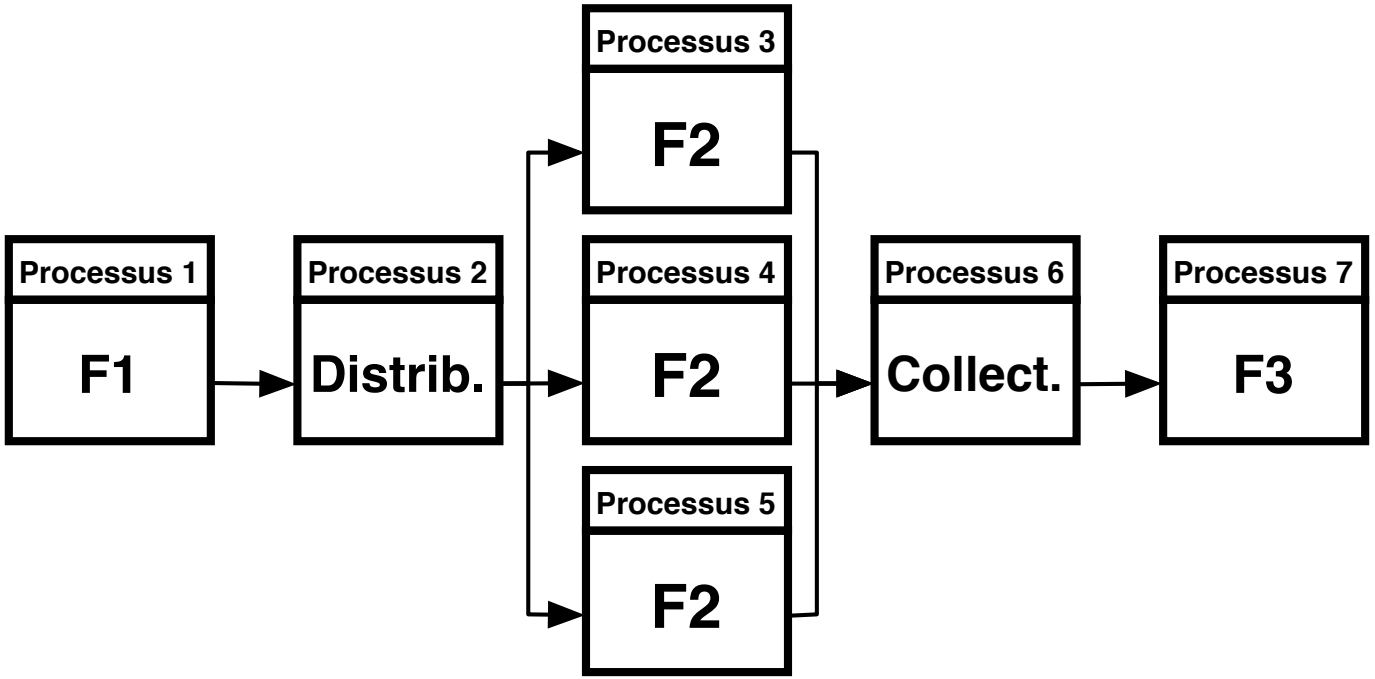
Principes

- Eléments récurrents de la programmation MIMD.
- Créer une application parallèle = combiner des squelettes.

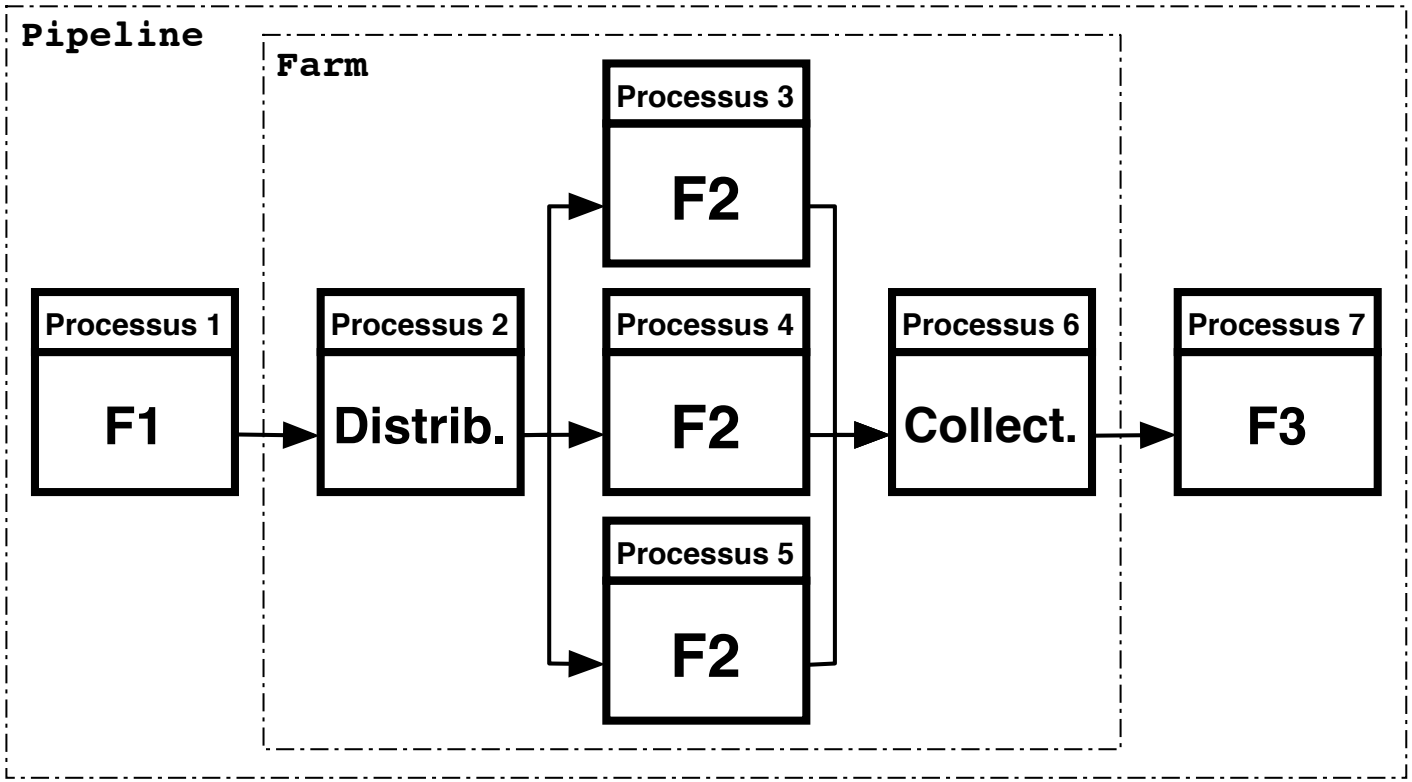
Mise en œuvre

- Définition d'un Graphe de Processus Communicants.
- Identification des structures récurrentes.
- Expression sous forme d'une combinaison des squelettes.

Les squelettes algorithmiques parallèles



Les squelettes algorithmiques parallèles



Les squelettes algorithmiques parallèles

Etat de l'Art

- C : eSkel [COLE 89]
- JAVA : Lithium [DANELUTTO 02]
- C++ : Muesli [KUCHEN 03]

Limitations

Les squelettes algorithmiques parallèles

Etat de l'Art

- C : eSkel [COLE 89]
- JAVA : Lithium [DANELUTTO 02]
- C++ : Muesli [KUCHEN 03]

Limitations

- Faible niveau d'abstraction.

Les squelettes algorithmiques parallèles

Etat de l'Art

- C : eSkel [COLE 89]
- JAVA : Lithium [DANELUTTO 02]
- C++ : Muesli [KUCHEN 03]

Limitations

- Faible niveau d'abstraction.
- Choix discutable des langages cibles.

Les squelettes algorithmiques parallèles

Etat de l'Art

- C : eSkel [COLE 89]
- JAVA : Lithium [DANELUTTO 02]
- C++ : Muesli [KUCHEN 03]

Limitations

- Faible niveau d'abstraction.
- Choix discutable des langages cibles.
- Technique d'implantation peu efficace.

Programmation du niveau SMP

Outils disponibles

- OpenMP [CLARK 98]
- pThread [IEEE POSIX 1003.1c]

Programmation du niveau SMP

Outils disponibles

- OpenMP [CLARK 98]
- pThread [IEEE POSIX 1003.1c]

Modèles de programmation

- OpenMP : boucle parallèle.
- pThread : processus explicites + sur-couche objet.

Programmation du niveau SIMD

L'extension AltiVec

- Extension SIMD de Motorola/IBM pour les PPC G4/G5.
- Accélération maximale théorique : $\times 4$ à $\times 16$.
- Interface de programmation : 162 primitives C.

Programmation du niveau SIMD

L'extension AltiVec

- Extension SIMD de Motorola/IBM pour les PPC G4/G5.
- Accélération maximale théorique : $\times 4$ à $\times 16$.
- Interface de programmation : 162 primitives C.

Outils existants

- Bibliothèque *Apple Accelerated Framework*
- BLAS/LAPACK
- Vectorisateur automatique de code : VAST

Programmer BABYLON

Constat

Hormis au niveau SMP, les outils existants pour programmer un *cluster* hybride ne répondent que partiellement à notre cahier des charges.

Programmer BABYLON

Constat

Hormis au niveau SMP, les outils existants pour programmer un *cluster* hybride ne répondent que partiellement à notre cahier des charges.

Objectifs

- Fournir une implantation C++ efficace du modèle squelette.
- Fournir une implantation C++ expressive des opérations SIMD.
- Garantir l'accessibilité de ces outils.

Première tentative : approche objet classique

Outil de programmation SIMD

- Classes encapsulant le concept de tableau numérique.
- Surcharges SIMD des opérateurs et fonctions classiques.
- Encapsulation des tâches annexes de vectorisation.

Première tentative : approche objet classique

Outil de programmation SIMD

- Classes encapsulant le concept de tableau numérique.
- Surcharges SIMD des opérateurs et fonctions classiques.
- Encapsulation des tâches annexes de vectorisation.

```
array<float> r(SZ),a(SZ),b(SZ),c(SZ);

r = cos(a)*(b+c);
```

Première tentative : approche objet classique

Outil de programmation MIMD

- Jeu de squelettes = Hiérarchie de classes.
- Fonctions séquentielles = Foncteurs.
- Combinaison de squelettes par polymorphisme de classe.

Première tentative : approche objet classique

Outil de programmation MIMD

- Jeu de squelettes = Hiérarchie de classes.
- Fonctions séquentielles = Foncteurs.
- Combinaison de squelettes par polymorphisme de classe.

```
class Smooth : public Task;
class Edge : public Task;
class Chain : public Task;

Farm* f      = new Farm(new Edge,3);
Pipeline* p = new Pipeline(new Smooth,f,new Chain);
p->run();
```

Résultats

Au niveau MIMD

- Temps d'exécution deux fois plus elevé.
- Pertes dues au polymorphisme de classe qui génère des accès mémoires supplémentaires lors de l'appel des méthodes virtuelles.

Résultats

Au niveau MIMD

- Temps d'exécution deux fois plus elevé.
- Pertes dues au polymorphisme de classe qui génère des accès mémoires supplémentaires lors de l'appel des méthodes virtuelles.

Au niveau SIMD

- Gains inférieurs à 30% du gain obtenu en C.
- La vectorisation automatique n'est pas toujours souhaitable.
- Pertes dues à la résolution dyadique des opérateurs.

Résolution dyadique des opérateurs

```
array<float> r(SZ),a(SZ),b(SZ),c(SZ);

r = cos(a)*(b+c);
```

Résolution dyadique des opérateurs

```
array<float> r(SZ),a(SZ),b(SZ),c(SZ);

array<float> tmp1(SZ);
for(int i=0;i<SZ;i++)
  tmp1[i] = cos(a[i]);
```

Résolution dyadique des opérateurs

```
array<float> r(SZ),a(SZ),b(SZ),c(SZ);

array<float> tmp1(SZ);
for(int i=0;i<SZ;i++)
  tmp1[i] = cos(a[i]);

array<float> tmp2(SZ);
for(int i=0;i<SZ;i++)
  tmp2[i] =b[i]+c[i];
```

Résolution dyadique des opérateurs

```
array<float> r(SZ),a(SZ),b(SZ),c(SZ);

array<float> tmp1(SZ);
for(int i=0;i<SZ;i++)
  tmp1[i] = cos(a[i]);

array<float> tmp2(SZ);
for(int i=0;i<SZ;i++)
  tmp2[i] = b[i]+c[i];

array<float> tmp3(SZ);
for(int i=0;i<SZ;i++)
  tmp3[i] = tmp1[i]*tmp2[i];
```

Résolution dyadique des opérateurs

```
array<float> r(SZ),a(SZ),b(SZ),c(SZ);

array<float> tmp1(SZ);
for(int i=0;i<SZ;i++)
  tmp1[i] = cos(a[i]);

array<float> tmp2(SZ);
for(int i=0;i<SZ;i++)
  tmp2[i] = b[i]+c[i];

array<float> tmp3(SZ);
for(int i=0;i<SZ;i++)
  tmp3[i] = tmp1[i]*tmp2[i];

for(int i=0;i<SZ;i++)
  r[i] = tmp3[i];
```

Résultats de l'approche classique

Constat

- Le polymorphisme et la surcharge des opérateurs contribuent à l'expressivité des outils mais grèvent sensiblement leur performance.
- Une grande partie du code ainsi généré est calculé à l'exécution alors qu'il est entièrement déterminé à la compilation.

Résultats de l'approche classique

Constat

- Le polymorphisme et la surcharge des opérateurs contribuent à l'expressivité des outils mais grèvent sensiblement leur performance.
- Une grande partie du code ainsi généré est calculé à l'exécution alors qu'il est entièrement déterminé à la compilation.

Solution

Évaluer le maximum de code possible à la compilation.

L'évaluation partielle

Principes

- Déterminer le code évaluable à la compilation.
- Appliquer des règles de transformation.
- Générer un code pré-optimisé.

L'évaluation partielle

Principes

- Déterminer le code évaluable à la compilation.
- Appliquer des règles de transformation.
- Générer un code pré-optimisé.

Solutions

- Modifier le compilateur.
- Utiliser des outils annexes.
- Utiliser les spécificités du langage hôte.

L'évaluation partielle

Principes

- Déterminer le code évaluable à la compilation.
- Appliquer des règles de transformation.
- Générer un code pré-optimisé.

Solutions

- Modifier le compilateur.
- Utiliser des outils annexes.
- Utiliser les spécificités du langage hôte.

L'évaluation partielle en C++

Éléments statiques du C++

- Les constantes entières.
- Les types natifs ou définis par l'utilisateur.
- Manipulables via les *templates*.

L'évaluation partielle en C++

Éléments statiques du C++

- Les constantes entières.
- Les types natifs ou définis par l'utilisateur.
- Manipulables via les *templates*.

La Méta-programmation *template*

- Méta-langage Turing-complet.
- Programmes exécutés à la compilation.
- Génération d'un code «recadré» effectivement compilé.

Applications

Outil de programmation SIMD

- Éliminer les copies superflues.
- Détecter les expressions vectorisables.
- Optimiser les boucles de traitements.

Applications

Outil de programmation SIMD

- Éliminer les copies superflues.
- Détecter les expressions vectorisables.
- Optimiser les boucles de traitements.

Outil de programmation MIMD

- Utiliser une version statique du polymorphisme de classe.
- Faciliter l'intégration du code existant.
- Optimiser les appels aux primitives de communications.

Applications

Outil de programmation SIMD

- Analyse de l'arbre de syntaxe abstraite (AST).
- Encodage de l'AST sous forme d'un type *template.*
- Marquage des éléments vectorisable de l'AST.

Applications

Outil de programmation SIMD

- Analyse de l'arbre de syntaxe abstraite (AST).
- Encodage de l'AST sous forme d'un type *template.*
- Marquage des éléments vectorisable de l'AST.

Outil de programmation MIMD

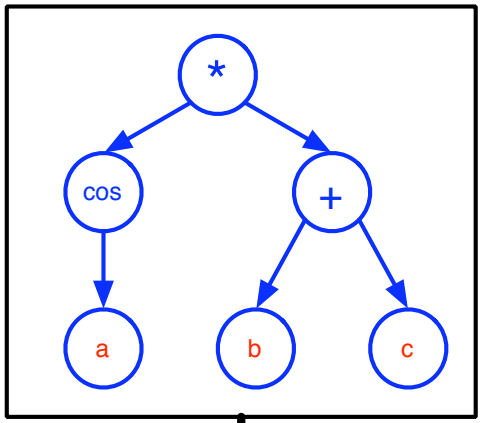
- Analyse du schéma de combinaison des squelettes.
- Encodage du schéma sous forme d'un type *template.*
- Optimisation du schéma [ALDINUCCI 99].
- Préparation des schémas de communications.

Applications

```
array<float> r,a(S),b(S),c(S);

r = cos(a)*(b+c);
```

Analyse statique



Méta-programmation template

```
node<Mul,node<Cos,array>,
          node<Plus,array,array>
    >
```

Génération de code

```
for(size_t i = 0;i <S;i++)
{
  r[i]=cos(a[i])*(b[i]+c[i]);
}
```

Conclusion

La méta-programmation *template*

- Permet d'implanter des mécanismes d'évaluation partielle en C++.
- Conserve l'expressivité des interfaces produites.
- Concept de *Bibliothèque Active*.

Conclusion

La méta-programmation *template*

- Permet d'implanter des mécanismes d'évaluation partielle en C++.
- Conserve l'expressivité des interfaces produites.
- Concept de *Bibliothèque Active*.

Contributions

- Programmation SIMD : E.V.E. (*SCPE 2005*).
- Programmation MIMD : QUAFF (*Parallel Computing 2005*).

La bibliothèque E.V.E.

Objectifs

Proposer une interface intuitive pour la manipulation de tableaux numériques et leur vectorisation via l'extension AltiVec.

La bibliothèque E.V.E.

Objectifs

Proposer une interface intuitive pour la manipulation de tableaux numériques et leur vectorisation via l'extension AltiVec.

Fonctionnalités

- Des classes de conteneurs génériques.
- Marquage local des conteneurs vectorisables.
- Large panel d'optimisations annexes.
- Syntaxe proche de MATLAB.

La bibliothèque E.V.E.

Code MATLAB

```
R = I(:,:,1);
G = I(:,:,2);
B = I(:,:,3);

Y = min(bitshift(abs(2104*R+4130*G+
                     802*B+135168),-13),235);
U = min(bitshift(abs(-1214*R-2384*G+
                      3598*B+1052672),-13),240);
V = min(bitshift(abs(3598*R-3013*G-
                     585*B+1052672),-13),240);
```

La bibliothèque E.V.E.

Code E.V.E.

```
array<int> R = I(All(),All(),0);
array<int> G = I(All(),All(),1);
array<int> B = I(All(),All(),2);

Y = min(shr(abs(2104*R+4130*G+
                802*B+135168),13),235);
U = min(shr(abs(-1214*R-2384*G+
                 3598*B+1052672),13),240);
V = min(shr(abs(3598*R-3013*G-
                585*B+1052672),13),240);
```

La bibliothèque E.V.E.

Code E.V.E. optimisé

```
view<int> R = I.ima_view(0);
view<int> G = I.ima_view(1);
view<int> B = I.ima_view(2);

Y = min(shr(abs(2104*R+4130*G+
                802*B+135168),13),235);
U = min(shr(abs(-1214*R-2384*G+
                 3598*B+1052672),13),240);
V = min(shr(abs(3598*R-3013*G-
                585*B+1052672),13),240);
```

La bibliothèque E.V.E.

Code E.V.E. optimisé SIMD

```
view<int,settings<simd> > R = I.ima_view(0);
view<int,settings<simd> > G = I.ima_view(1);
view<int,settings<simd> > B = I.ima_view(2);

Y = min(shr(abs(2104*R+4130*G+
                802*B+135168),13),235);
U = min(shr(abs(-1214*R-2384*G+
                 3598*B+1052672),13),240);
V = min(shr(abs(3598*R-3013*G-
                585*B+1052672),13),240);
```

La bibliothèque E.V.E.

Performances brutes

- Mode scalaire : 80-90% de la vitesse d'exécution en C.
- Mode vectoriel
  - Fonctions natives : 50-75% du gain maximum.
  - Fonctions composites : 70-95% du gain maximum.

La bibliothèque E.V.E.

Performances brutes

- Mode scalaire : 80-90% de la vitesse d'exécution en C.
- Mode vectoriel
  - Fonctions natives : 50-75% du gain maximum.
  - Fonctions composites : 70-95% du gain maximum.

Quelques applications

Test	AltiVec	Naïf	E.V.E.
$\sum a[i]$	15.7	8.0	15.4
Produit matriciel	4.8	1.4	4.6
Différence d'image	7.8	2.3	7.7
Filtrage Gaussien	7.9	0.3	7.8
Harris & Stephen	4.9	0.1	4.8

La bibliothèque E.V.E.

Performances brutes

- Mode scalaire : 80-90% de la vitesse d'exécution en C.
- Mode vectoriel
  - Fonctions natives : 50-75% du gain maximum.
  - Fonctions composites : 70-95% du gain maximum.

Quelques applications

Test	AltiVec	Naïf	E.V.E.
$\sum a[i]$	15.7	8.0	15.4
Produit matriciel	4.8	1.4	4.6
Différence d'image	7.8	2.3	7.7
Filtrage Gaussien	7.9	0.3	7.8
Harris & Stephen	4.9	0.1	4.8

la bibliothèque QUAFF

Objectif

Fournir une implantation efficace et intuitive pour la spécification d'applications parallèles via des squelettes algorithmiques..

la bibliothèque QUAFF

Objectif

Fournir une implantation efficace et intuitive pour la spécification d'applications parallèles via des squelettes algorithmiques..

Fonctionnalités

- Intégration non-intrusives des fonctions utilisateurs.
- Jeu de squelettes classique :

la bibliothèque QUAFF

Objectif

Fournir une implantation efficace et intuitive pour la spécification d'applications parallèles via des squelettes algorithmiques..

Fonctionnalités

- Intégration non-intrusives des fonctions utilisateurs.
- Jeu de squelettes classique :
  - Parallélisme de tâches : `pipeline`, `pardo`

la bibliothèque QUAFF

Objectif

Fournir une implantation efficace et intuitive pour la spécification d'applications parallèles via des squelettes algorithmiques..

Fonctionnalités

- Intégration non-intrusives des fonctions utilisateurs.
- Jeu de squelettes classique :
  - Parallélisme de tâches : `pipeline`, `pardo`
  - Parallélisme de données : `farm`, `scm`.

la bibliothèque QUAFF

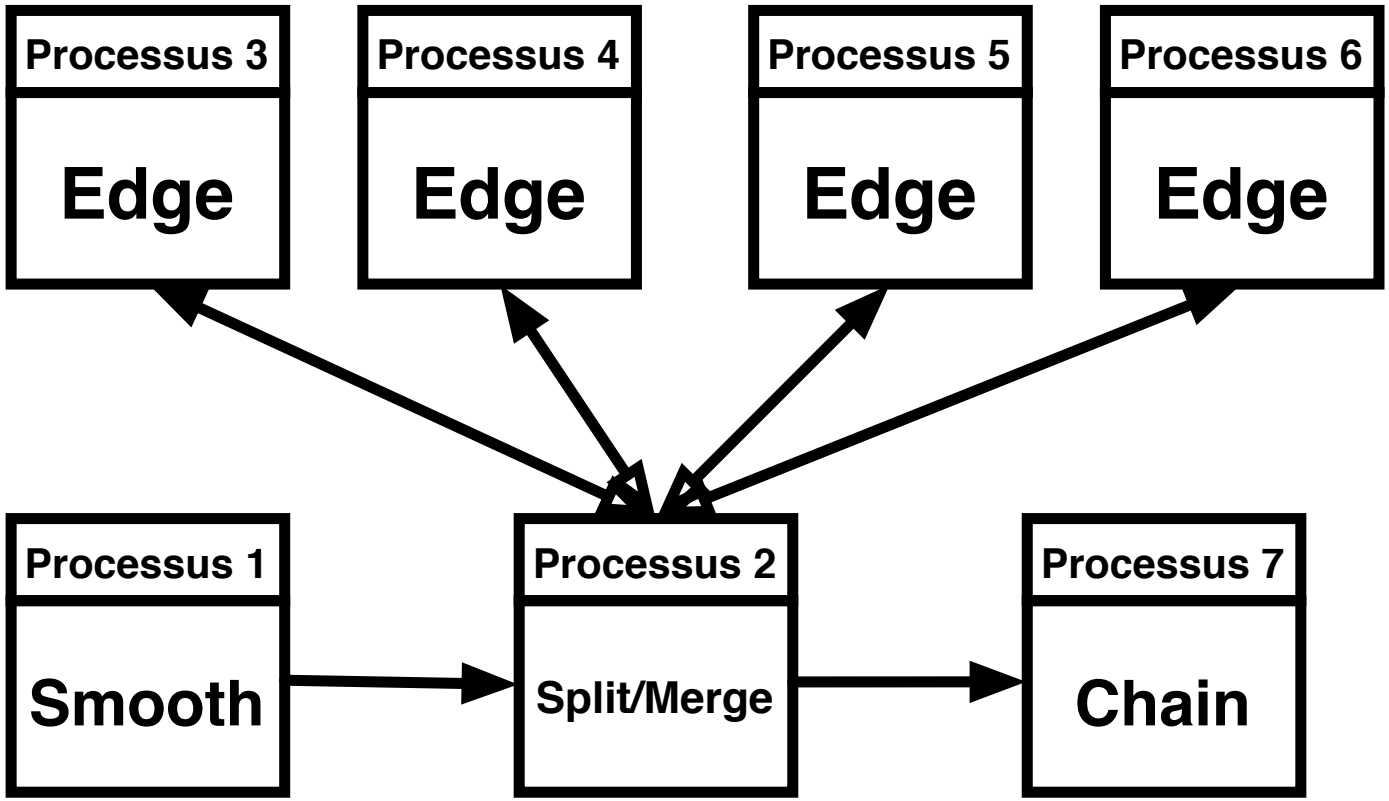
Objectif

Fournir une implantation efficace et intuitive pour la spécification d'applications parallèles via des squelettes algorithmiques..

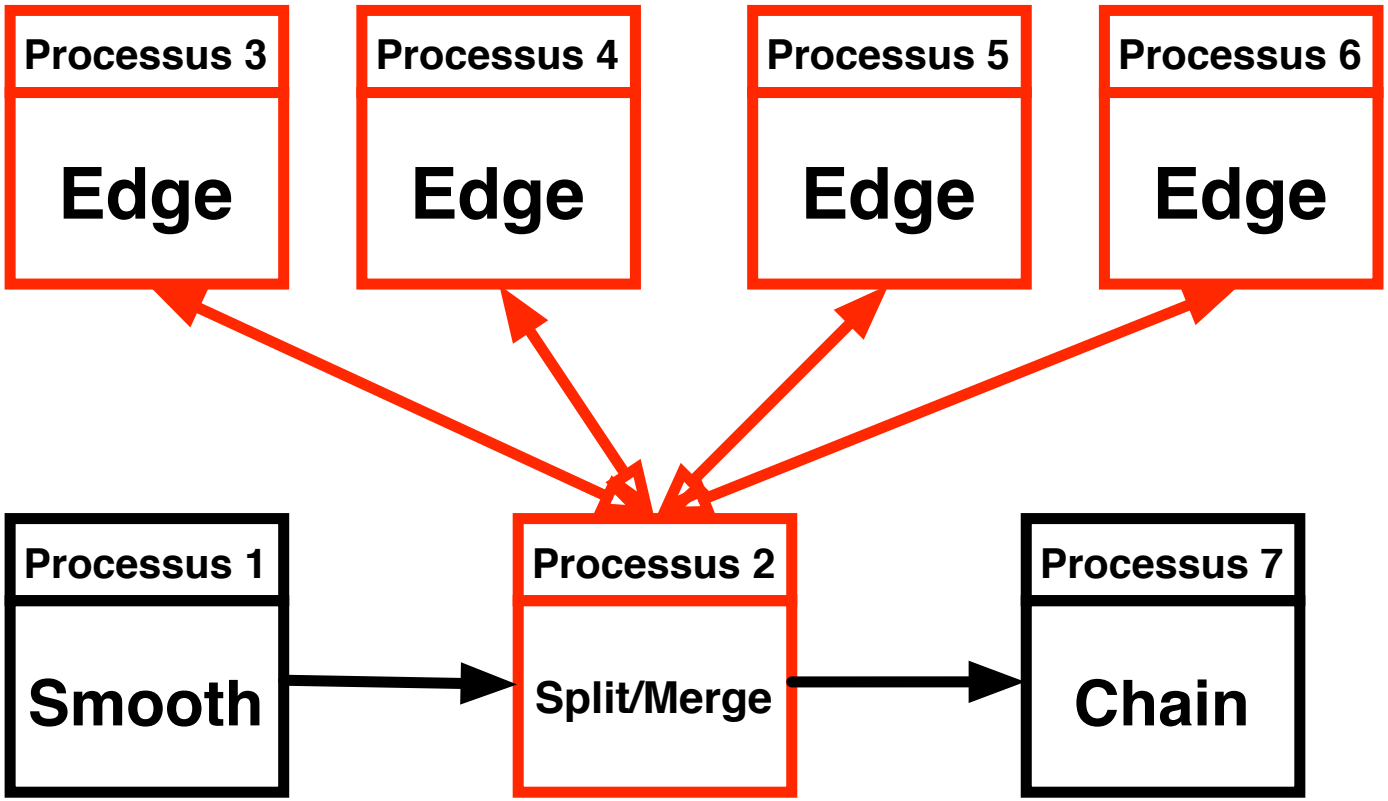
Fonctionnalités

- Intégration non-intrusives des fonctions utilisateurs.
- Jeu de squelettes classique :
  - Parallélisme de tâches : `pipeline`, `pardo`
  - Parallélisme de données : `farm`, `scm`.
  - Squelettes séquentiels : `sequence`, `select`.

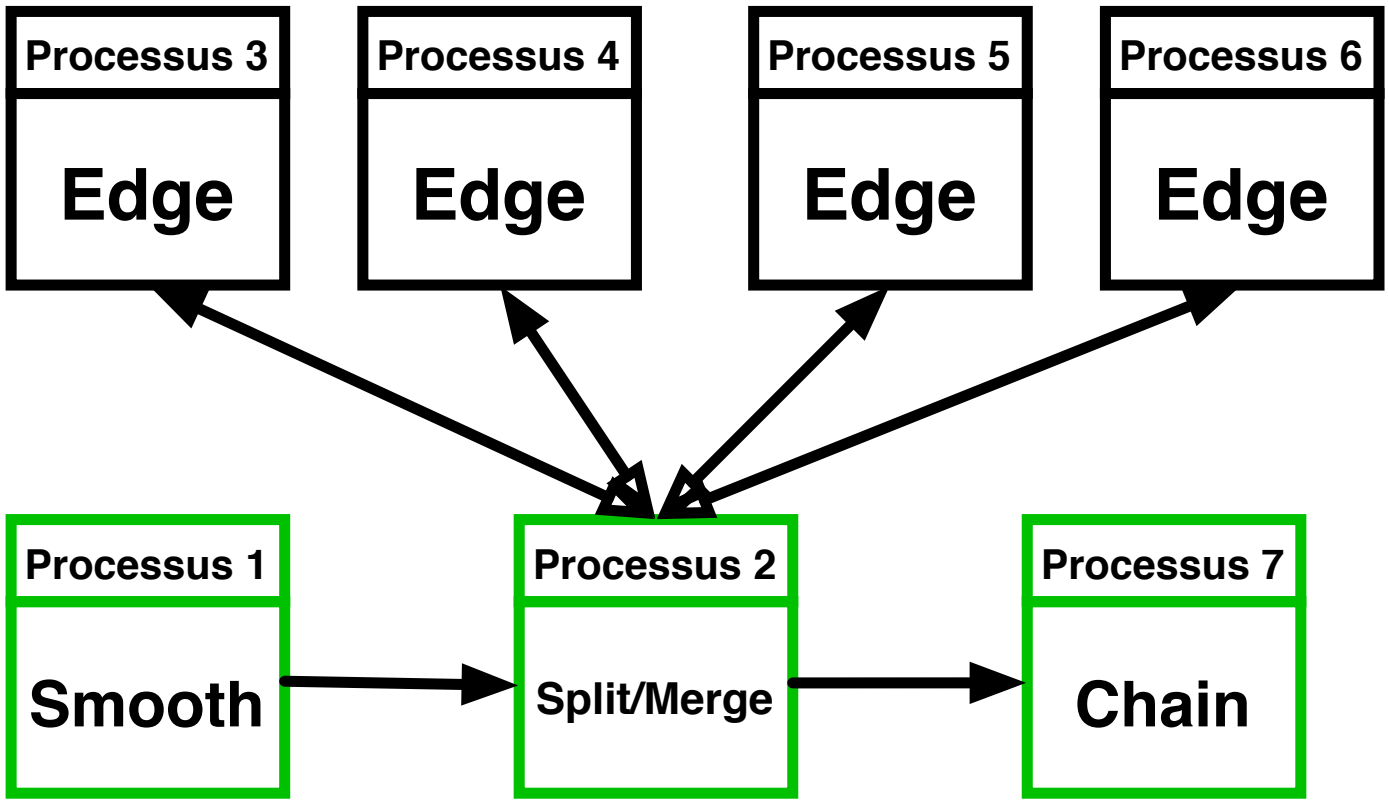
La bibliothèque QUAFF – Exemple



La bibliothèque QUAFF – Exemple



La bibliothèque QUAFF – Exemple



La bibliothèque QUAFF – Exemple

Code QUAFF

```
typedef task<Smooth,none_t,Image>              smooth;
typedef task<Split,Image,Image>                 split;
typedef task<Merge,Image,Image>                 merge;
typedef task<Edge,Image,vector<Line>>            edge;
typedef task<Chain,vector<Line>,none_t>         chain;
```

La bibliothèque QUAFF – Exemple

Code QUAFF

```
typedef task<Smooth,none_t,Image>                smooth;
typedef task<Split,Image,Image>                   split;
typedef task<Merge,Image,Image>                   merge;
typedef task<Edge,Image,vector<Line>>              edge;
typedef task<Chain,vector<Line>,none_t>           chain;

typedef scm<split,worker<edge,4>,merge>         process;
```

La bibliothèque QUAFF – Exemple

Code QUAFF

```
typedef task<Smooth,none_t,Image>                smooth;
typedef task<Split,Image,Image>                   split;
typedef task<Merge,Image,Image>                   merge;
typedef task<Edge,Image,vector<Line>>              edge;
typedef task<Chain,vector<Line>,none_t>           chain;

typedef scm<split,worker<edge,4>,merge>         process;
typedef pipeline<stage<smooth,process,chain>>     app;
```

La bibliothèque QUAFF – Exemple

Code QUAFF

```
typedef task<Smooth,none_t,Image>                smooth;
typedef task<Split,Image,Image>                   split;
typedef task<Merge,Image,Image>                   merge;
typedef task<Edge,Image,vector<Line>>              edge;
typedef task<Chain,vector<Line>,none_t>           chain;

typedef scm<split,worker<edge,4>,merge>       process;
typedef pipeline<stage<smooth,process,chain>>  app;

application<app> detector;
detector.run();
```

La bibliothèque QUAFF

Performances

- Surcoût sur les communications $\approx$ 0.5%
- Surcoût sur les squelettes $\approx$ 1%
- Surcoût sur les application $\approx$ 4%

La bibliothèque QUAFF

Performances

- Surcoût sur les communications $\approx 0.5\%$
- Surcoût sur les squelettes $\approx 1\%$
- Surcoût sur les application $\approx 4\%$

Pertinence [COLE 99]

- Diffusion du concept.
- Support du parallélisme *ad hoc*.
- Quel gain en terme de temps de développement ?

Sommaire

Le suivi de piéton

Objectifs

Détecter et suivre un piéton présent dans une séquence vidéo stéréoscopique à la cadence du flux.

Le suivi de piéton

Objectifs

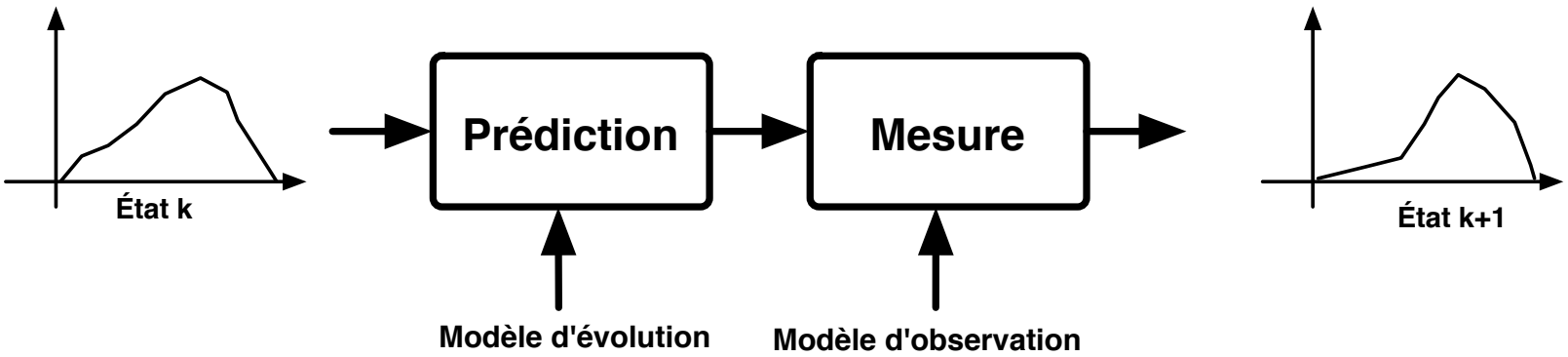
Détecter et suivre un piéton présent dans une séquence vidéo stéréoscopique à la cadence du flux.

Difficultés

- Variations d'apparence, de pose, d'éclairement.
- Occultation partielle.
- Arrière plan non-statique.

Le suivi d'objet - Approche probabiliste

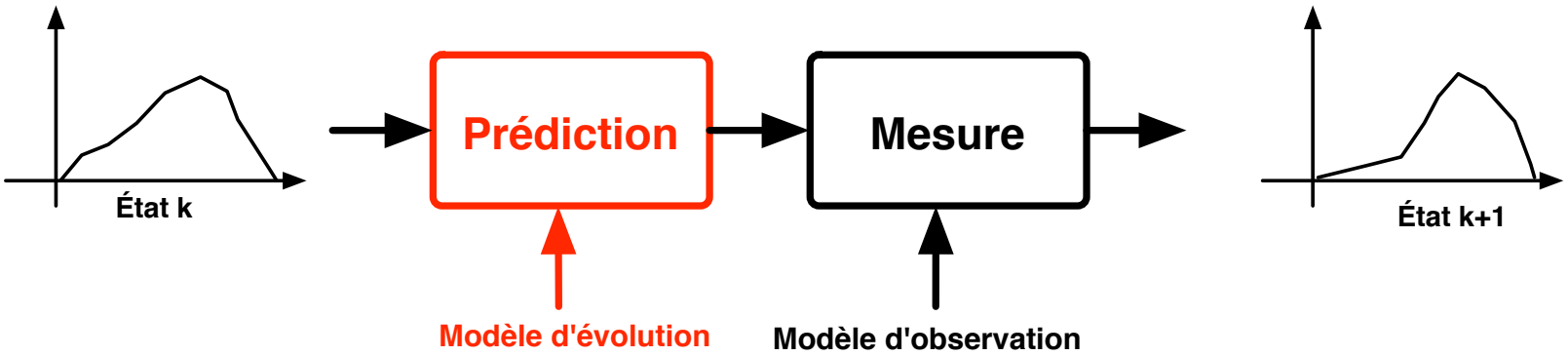
Principes



Le suivi d'objet - Approche probabiliste

Principes

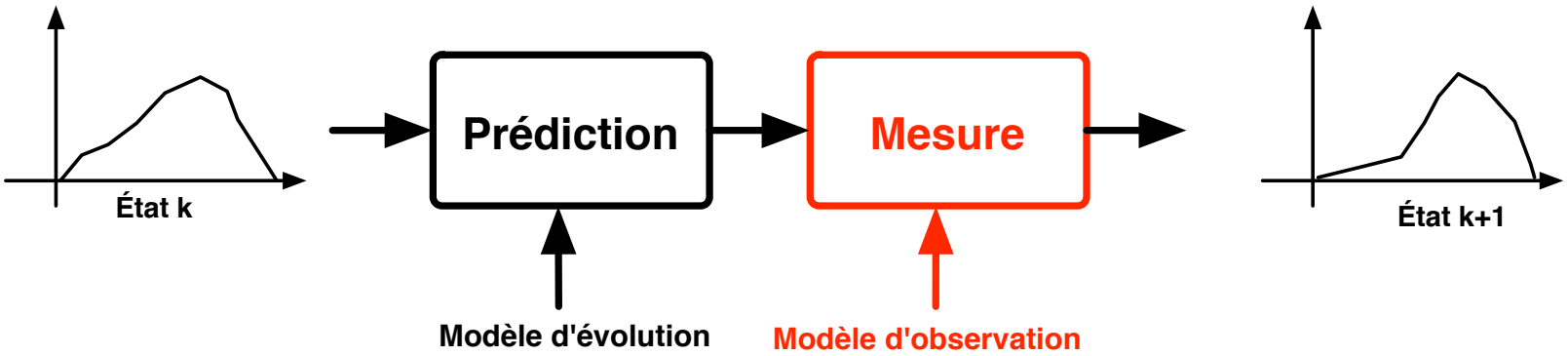
- Prédire la position de l'objet.
- Effectuer une mesure
- Estimer la nouvelle position



Le suivi d'objet - Approche probabiliste

Principes

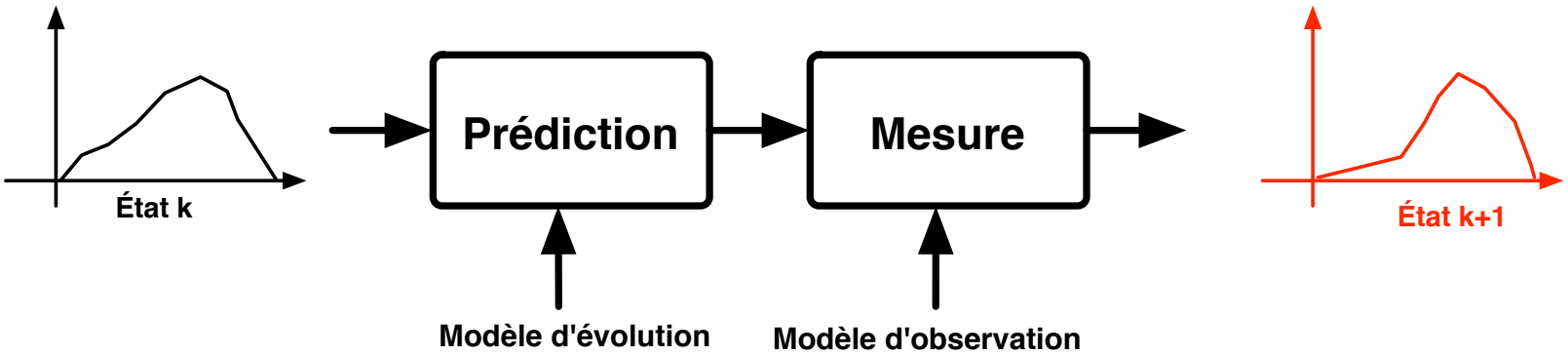
- Prédire la position de l'objet.
- Effectuer une mesure
- Estimer la nouvelle position



Le suivi d'objet - Approche probabiliste

Principes

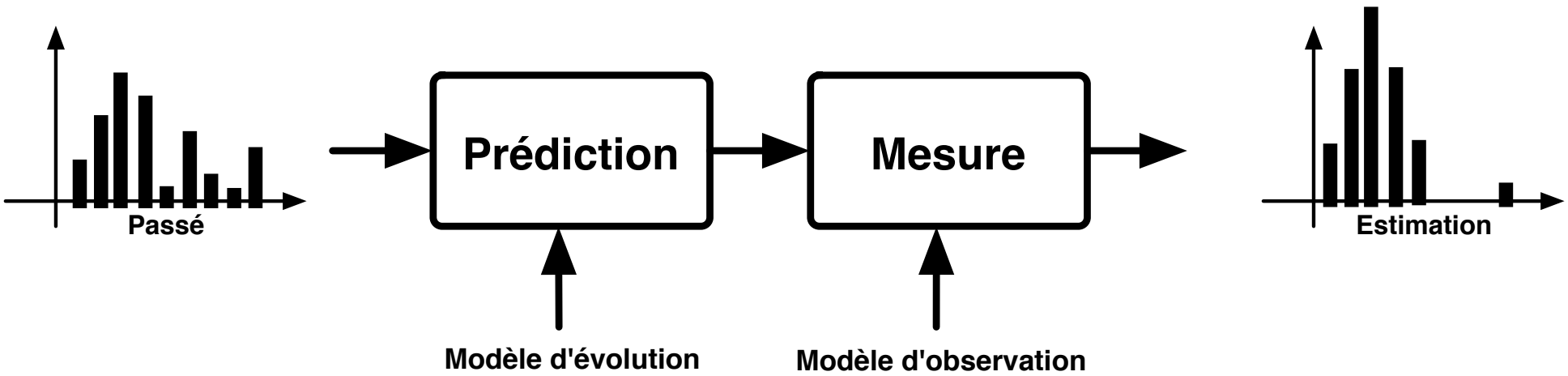
- Prédire la position de l'objet.
- Effectuer une mesure
- Estimer la nouvelle position



Filtrage particulaire [BLAKE 96]

Mise en œuvre

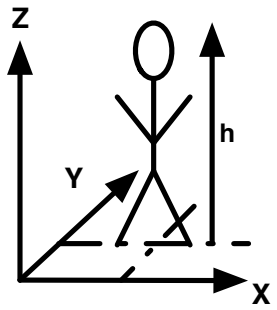
- Un modèle d'état.
- Un modèle d'évolution.
- Un modèle d'observation (fonction de vraisemblance).



Filtrage particulaire [BLAKE 96]

Modèle d'état

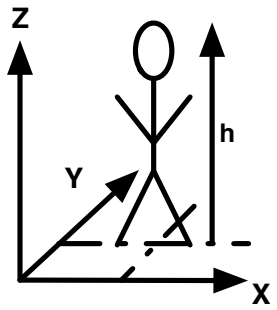
$$\mathbf{X}_t = (x_t, y_t, z_t, h_t)$$



Filtrage particulaire [BLAKE 96]

Modèle d'état

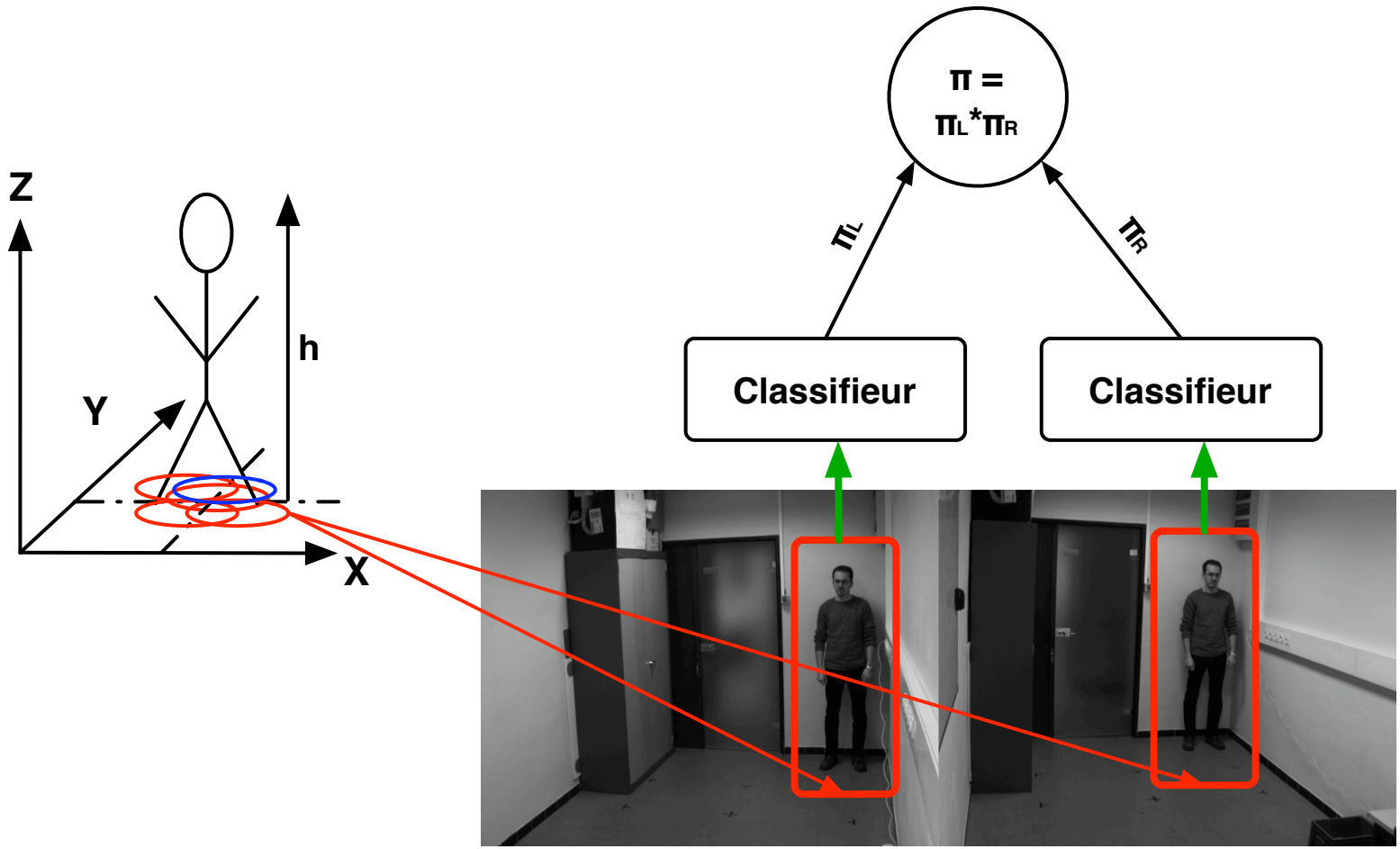
$$\mathbf{X}_t = (x_t, y_t, z_t, h_t)$$



Modèle d'évolution

$$\mathbf{X}_{t+1} = \mathbf{A}\mathbf{X}_t + \mathbf{B}\mathbf{v}_t, \ \mathbf{v}_t \sim \mathcal{N}(0, \Sigma)$$

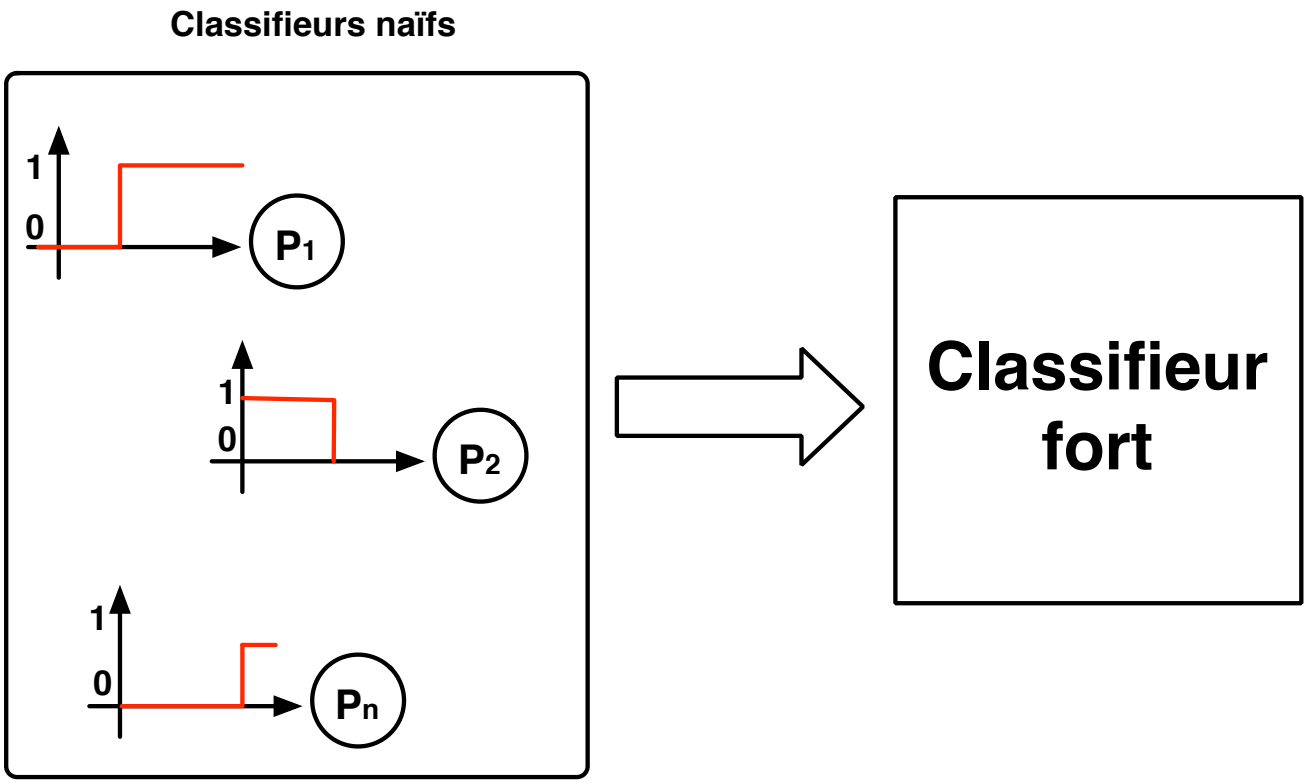
Filtrage particulaire - Modèle d'observation



Le Boosting

Principe [FREUND 95]

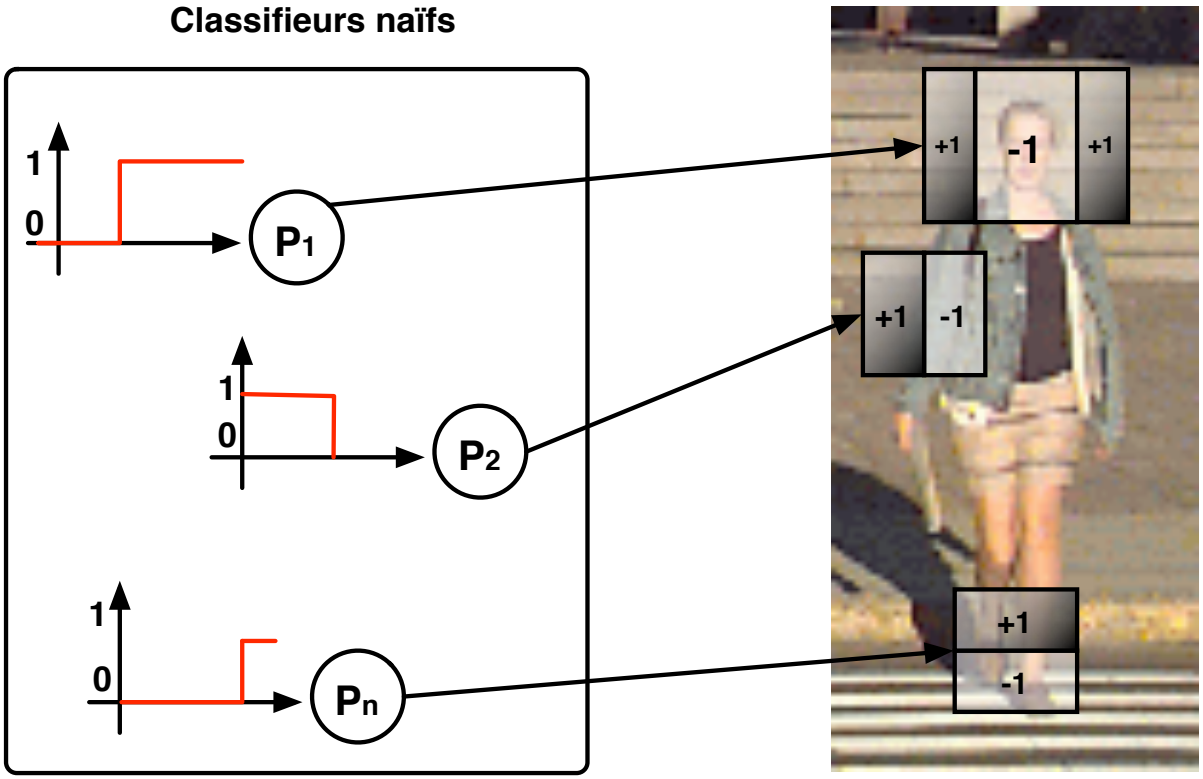
Combiner des classifieurs «naïfs» afin de composer un classifieur performant.



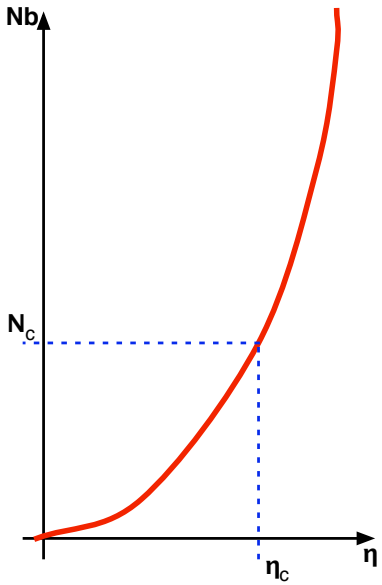
Le Boosting - Descripteur d'images

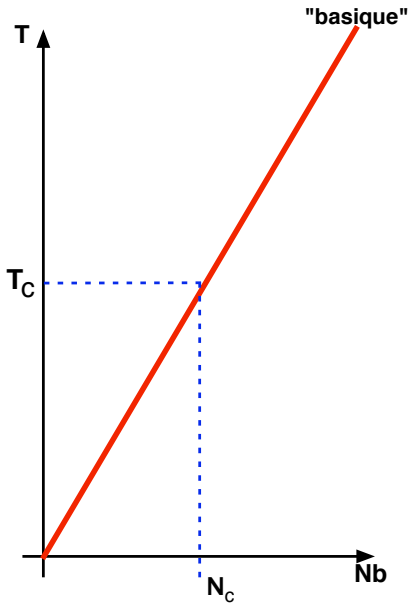
Principe [PAPAGEORGIOU 97]

Classifieur «naïf» = seuil sur la réponse à une ondelette de Haar.

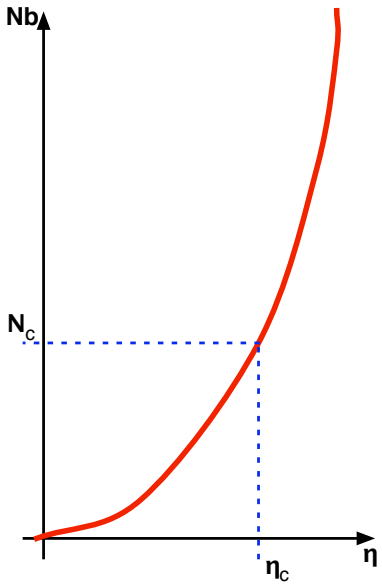


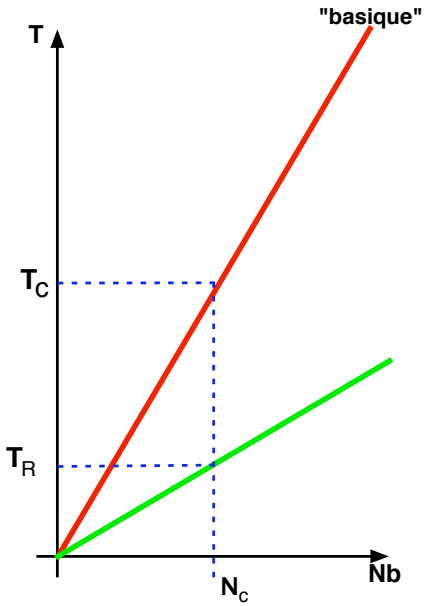
Filtrage particulaire - Intérêt de la parallélisation



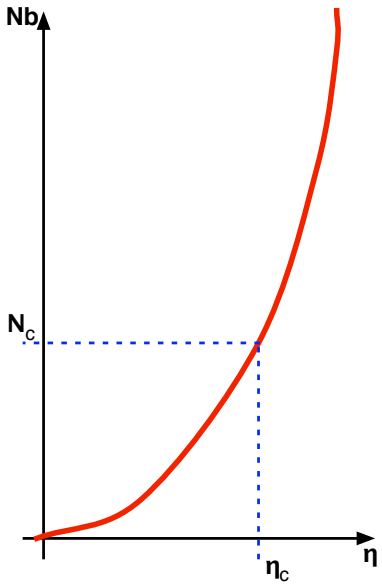


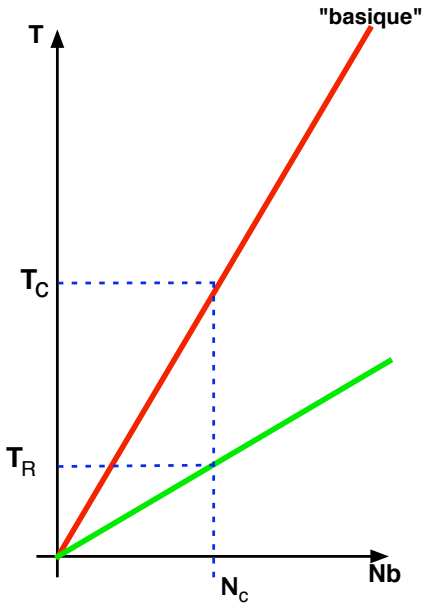
Filtrage particulaire - Intérêt de la parallélisation





Filtrage particulaire - Intérêt de la parallélisation

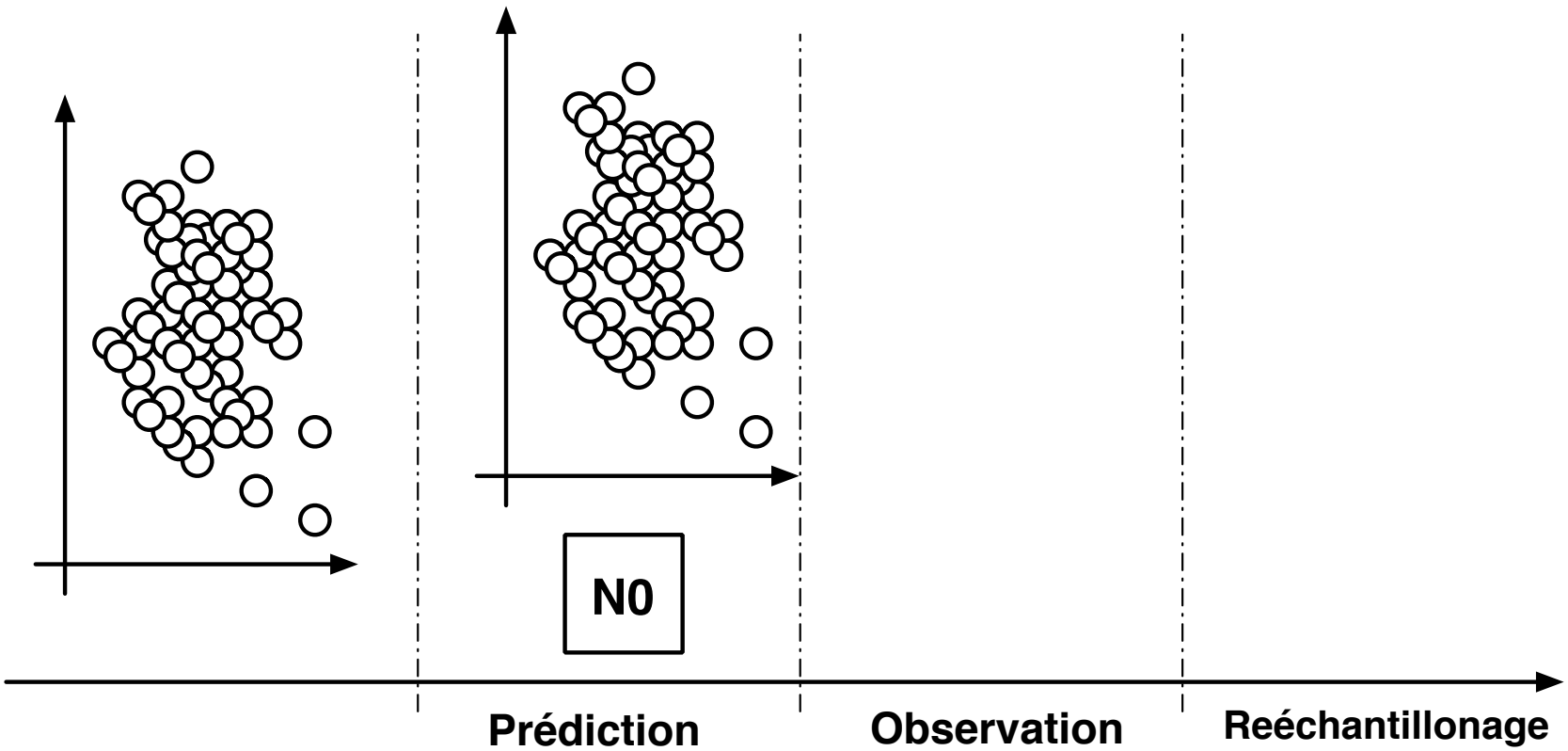




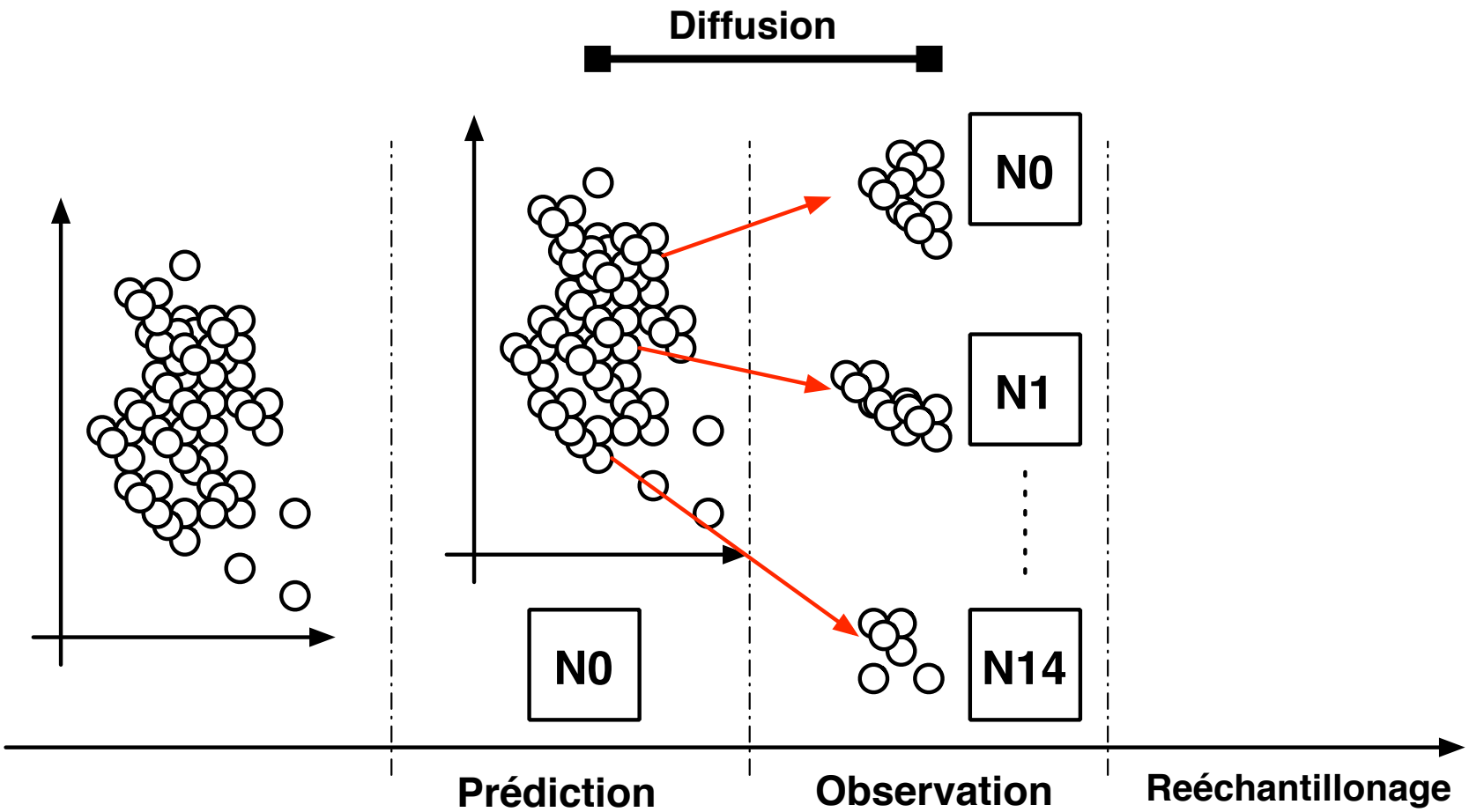
Constat

La phase d'observation représente 80% du temps d'exécution. Nous avons focaliser les efforts de parallélisation sur la fonction d'observation.

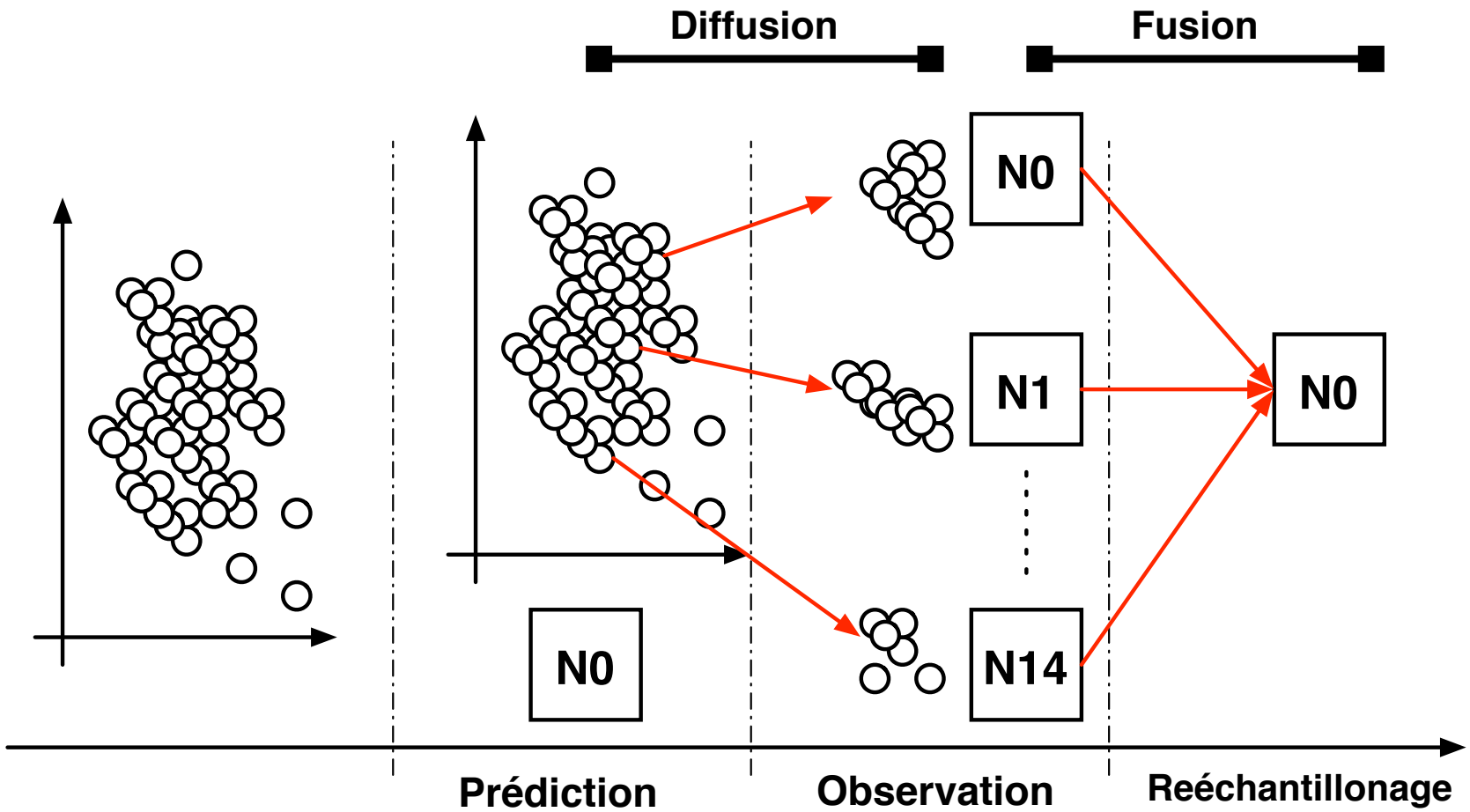
Suivi de piéton - Parallélisation



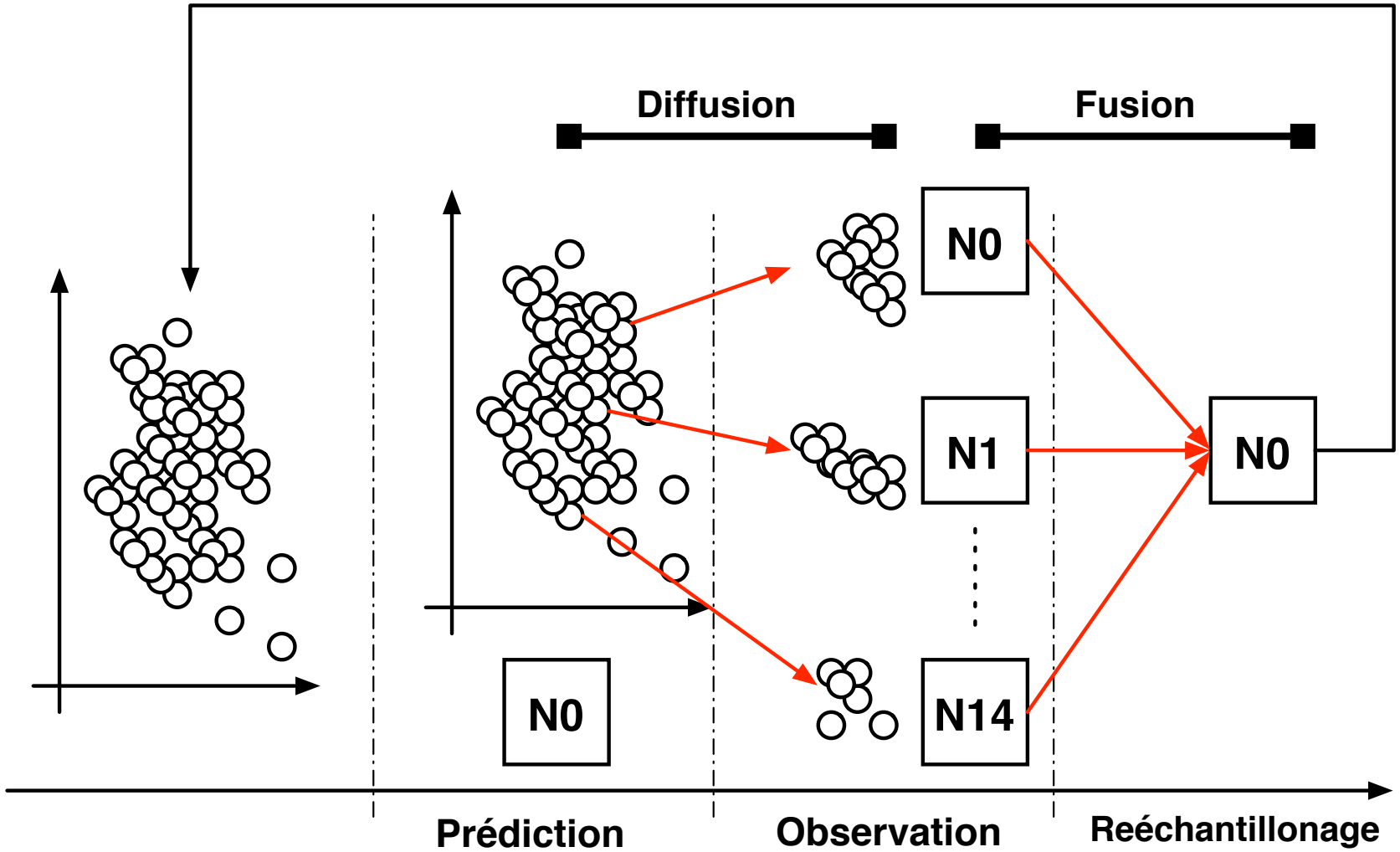
Suivi de piéton - Parallélisation



Suivi de piéton - Parallélisation



Suivi de piéton - Parallélisation



Suivi de piéton - Parallélisation

Apport du SMP

- Mesures indépendantes dans chacune des images.
- Fusion des poids avant transmission au nœud principal.

Suivi de piéton - Parallélisation

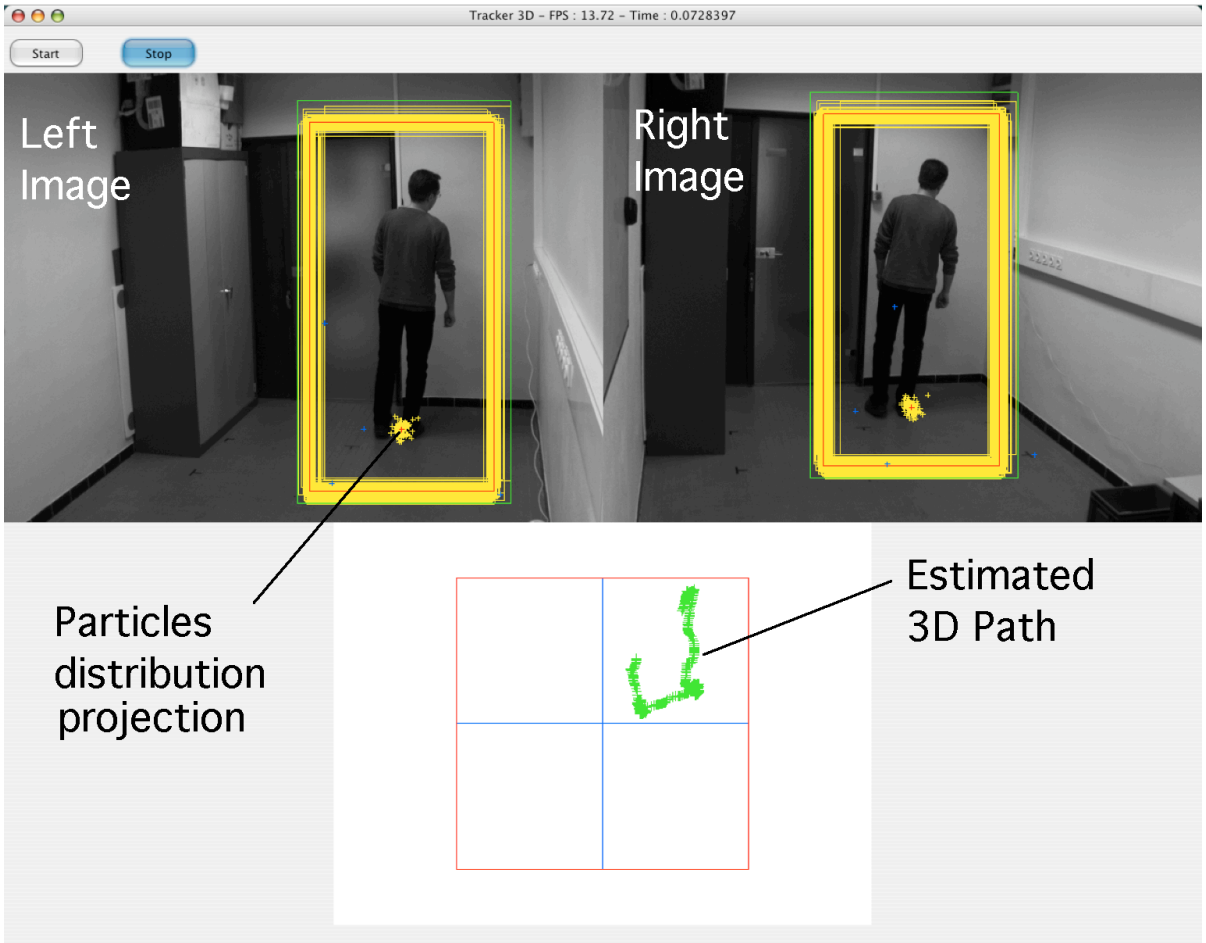
Apport du SMP

- Mesures indépendantes dans chacune des images.
- Fusion des poids avant transmission au nœud principal.

Apport du SIMD

- Vectorisation du modèle d'évolution.
- Vectorisation des calculs nécessaires aux projections.

Suivi de piéton - Application



Suivi de piéton - Performances

	100	500	1000	5000
Base	0.061s	0.299s	0.704s	11.358
Fréquence	16.38	3.34	1.42	0.09
BABYLON	0.014s	0.0195s	0.0334s	0.105s
Fréquence	70.8	51.3	29.96	9.4
Gain	4.31	15.38	21.1	107.35

Suivi de piéton - Performances

	100	500	1000	5000
Base	0.061s	0.299s	0.704s	11.358
Fréquence	16.38	3.34	1.42	0.09
BABYLON	0.014s	0.0195s	0.0334s	0.105s
Fréquence	70.8	51.3	29.96	9.4
Gain	4.31	15.38	21.1	107.35

Résultats (CIMCV/ECCV 06)

- Temps réel atteint pour 1000 particules.

Suivi de piéton - Performances

	100	500	1000	5000
Base	0.061s	0.299s	0.704s	11.358
Fréquence	16.38	3.34	1.42	0.09
BABYLON	0.014s	0.0195s	0.0334s	0.105s
Fréquence	70.8	51.3	29.96	9.4
Gain	4.31	15.38	21.1	107.35

Résultats (CIMCV/ECCV 06)

- Temps réel atteint pour 1000 particules.
- Gain maximum de l'ordre de $\times 100$.

Sommaire

La Machine BABYLON

Une architecture innovante

- Double bus de communications.
- *Cluster* hybride MIMD/SMP/SIMD.
- Support à des applications réalistes.
- Performances : gain de x30 à x100.

La Machine BABYLON

Une architecture innovante

- Double bus de communications.
- *Cluster* hybride MIMD/SMP/SIMD.
- Support à des applications réalistes.
- Performances : gain de x30 à x100.

Des outils pertinents

- E.V.E. : Interface expressive pour les extensions SIMD.
- QUAFF : Implantation performante du modèle squelette.

Perspectives

Au-delà de BABYLON

- Calculs déportés.
- Ouverture hors de la vision.

Perspectives

Au-delà de BABYLON

- Calculs déportés.
- Ouverture hors de la vision.

Au-delà de E.V.E. et QUAFF

- Interactions entre les bibliothèques ?
- QUAFF sur la Grille ?
- E.V.E. pour SSE2 ?
- Des méta-outils de méta-programmation ?

Merci de votre attention.