Christophe Collet

Principes et Technologies des Architectures Spécialisées

Introduction

Introduction

Pourquoi des architectures spécialisées ?

- Certains systèmes ont besoin de fonctionnalités sous certaines contraintes qui ne peuvent être respectées si ces fonctions sont exécutées par un processeur

- Deux grandes contraintes :
  - Temps de réponse ou de traitement
  - Consommation énergétique

Introduction

Des mécanismes matériels adaptés à certains algorithmes

- Pour un meilleur rapport performance/énergie
  - Décaleur+ALU
  - Arithmétique saturée
  - MAC : *Multiply and accumulate*

- Pour plus de performance
  - SIMD/GPU

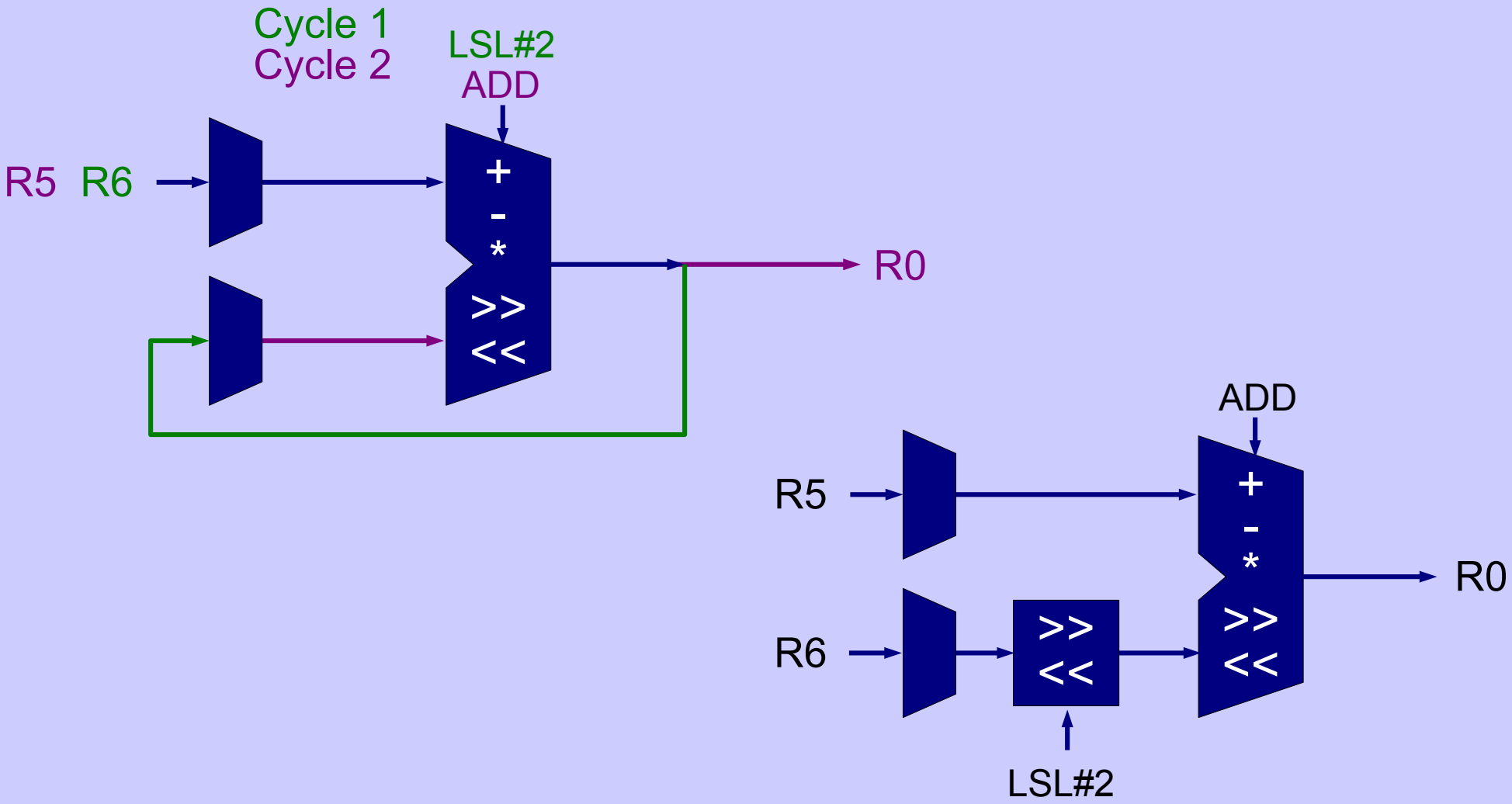
Introduction

Mécanismes en voie d'intégration

- Certaines idées des architectures spécialisées sont intégrées dans les processeurs généralistes
  - SIMD, MAC, arithmétique saturée
- Mais
  - Certaines applications nécessitent une performance telle qu'un processeur généraliste ne peut la satisfaire
  - Pour de très grandes séries, il est encore parfois plus rentable d'avoir des circuits spécialisés ou configurables

Décaleur + ALU

- **ADD RO,[R5,R6 LSL#2]** @ R0←add(R5+R6<<2)



Arithmétique saturée

Exemple : éclaircir une image

```
#define BRIGHTENING_CONSTANT 0x64
unsigned char bitmap[BITMAP_SIZE];
size_t i;
/* Charger l'image... */
for(i = 0; i < BITMAP_SIZE; i ++)
   bitmap[i] += BRIGHTENING_CONSTANT;
```

250	11111010
+ 100	+ 01100100
350	= 101011110
94	= 01011110

Overflow !

MAC : *Multiply and accumulate*

```
For i=1 to N
{
  update j, update k
  a ← a + x_j y_k
}
```

ARM : opération MLA

Utilisé dans le produit scalaire de vecteurs,

$$a = \sum_{k=0}^{n} x_k \, y_k$$

la convolution, les équations différentielles,
les transformées de Fourier,...

Quelles applications

- Traitement de données numériques fournies par des capteurs ou générées à partir de modèles
  - Multimédia : son, voix, image, vidéo...
    transmission, stockage, synthèse...
    → compression, restauration, amélioration, filtrage...
  - Sécurité, tous types de données
    → cryptage
  - Automatique : robotique, automobile, sécurité industrielle...
    → traitement du signal temps réel

Introduction

- Problèmes spécifiques au traitement d'images numériques :
  - Taille des données : stockage en mémoires - cache, centrale et périphériques
  - Temps de traitement : accès et opérations sur chaque pixel
  - Fréquence pour la vidéo : transfert entre composants via les bus

Taille des images

- exemples :
  - 1Mp : 1024x1024 pixels x 3 couleurs = 3 Mo
  - 5Mp : 2592x1944 pixels x 3 couleurs = 14 Mo
  - 12,8Mp : 4368x2912 pixels x 3 couleurs = 36,4 Mo
  - 21Mp : 5184x3456 pixels x 3 couleurs = 51,2 Mo
  - 100Mp : = 300 Mo
  - ...
- Aujourd'hui : mémoire centrale des machines >> 1 Go
- problème dû au cache *(L2 entre 256 et 512 Ko / core)*
  → nombreux accès à la mémoire centrale

Temps de traitement

- Complexité - image de *n* pixels :
  - Histogramme : $O(n)$
  - filtrage, convolution : $O(n*m^2)$
  - Hough (droite) : $O(n^3)$
  - Fourier (FFT) : $O(n \log n)$
- Calcul sur flottants : 1 à 2 instructions par cycle

  → jusqu'à plusieurs jours par image

Fréquence vidéo

- exemples :
  - VGA : 640x480 px $*$ 3c $*$ 30 im/s = 26,3 Mo/s
  - vidéo PAL : 768x575 px $*$ 3c $*$ 25 im/s = 31,6 Mo/s
  - HD 720p : 1280x720 px $*$ 3c $*$ 60 im/s = 158 Mo/s
  - HD 1080p : 1920x1080 px $*$ 3c $*$ 30 im/s = 177 Mo/s
  - 1Mp : 1024x1024 px $*$ 3c $*$ 100 im/s = 300 Mo/s
  - HD 4K : 4096x2160 px $*$ 3c $*$ 30 im/s = 760 Mo/s
  - Stéréo $\rightarrow$ $\times 2$

Fréquence vidéo

- bus externes :
  - Usb 3.2 : 20 Gbit/s
  - firewire ieee1394 : 50 Mo/s (100 Mo/s *version b*)
  - Camera Link : 230 Mo/s ∗ 1,2 ou 3
  - Thunderbolt 4 : 40 Gbit/s (2 canaux)
- interface numérique :
  - HDMI v2.1 : 48 Gbit/s

Fréquence vidéo

- bus internes :
  - PCI express : ~1 Go/s *(v3.0 par lien, jusqu'à 16 liens)*
    ~2 Go/s *(v4.0)*
  - autres : bus système, bus mémoire... plusieurs Go/s
- écriture sur disque : 100 à 200 Mo/s au mieux *(mode soutenu)*
  400 à 450 Mo/s *avec des SSD.*
  mais à saturation du cache les performances chutent !

  → utilisation du raid 0 : plusieurs disques en parallèle
- traitement en temps-réel de la vidéo :
  - 25, 30, 50, 60 im/s voir plus !

Quelle configuration ?

- Selon les besoins applicatifs :
  - ordinateur de configuration moyenne suffisant pour les besoins courant : traitement et stockage photo et vidéo.
  - serveur raid 0 pour la production, le stockage et la diffusion en vidéo et images haute résolution.
  - matériel spécifique pour les traitements des images et de la vidéo en un temps *acceptable* (temps-réel, 1s, 1mn, 1h, 1j ...) pour des applications industrielles

Quelle configuration ?

- Mais :

Traitements rapides

⇒

Architectures spécifiques

- machines séquentielles montrent rapidement leurs limites

  donc parallélisation logicielle et matérielle

Parallélisation : Introduction

- Motivation du parallélisme :
  - Puissance de calcul :
    - Limitation technologique des machines séquentielles (type Von Neumann).
    - Le parallélisme est introduit d'abord dans les processeurs, puis dans les machines.
  - Coût :
    - rapport coût/performances
    - accroissement de la puissance d'un proc = cher
    - meilleur rapport pour la multiplication des proc

Parallélisation : notion d'accélération

- Accélération = gain de temps obtenu lors de la parallélisation du programme séquentiel.

- Définition : Soit ***T1*** le temps nécessaire à un programme pour résoudre le problème A sur un ordinateur séquentiel et soit ***Tp*** le temps nécessaire à un programme pour résoudre le même problème A sur un ordinateur parallèle contenant ***p*** processeurs, alors l'accélération (*Speed-Up*) est le rapport :

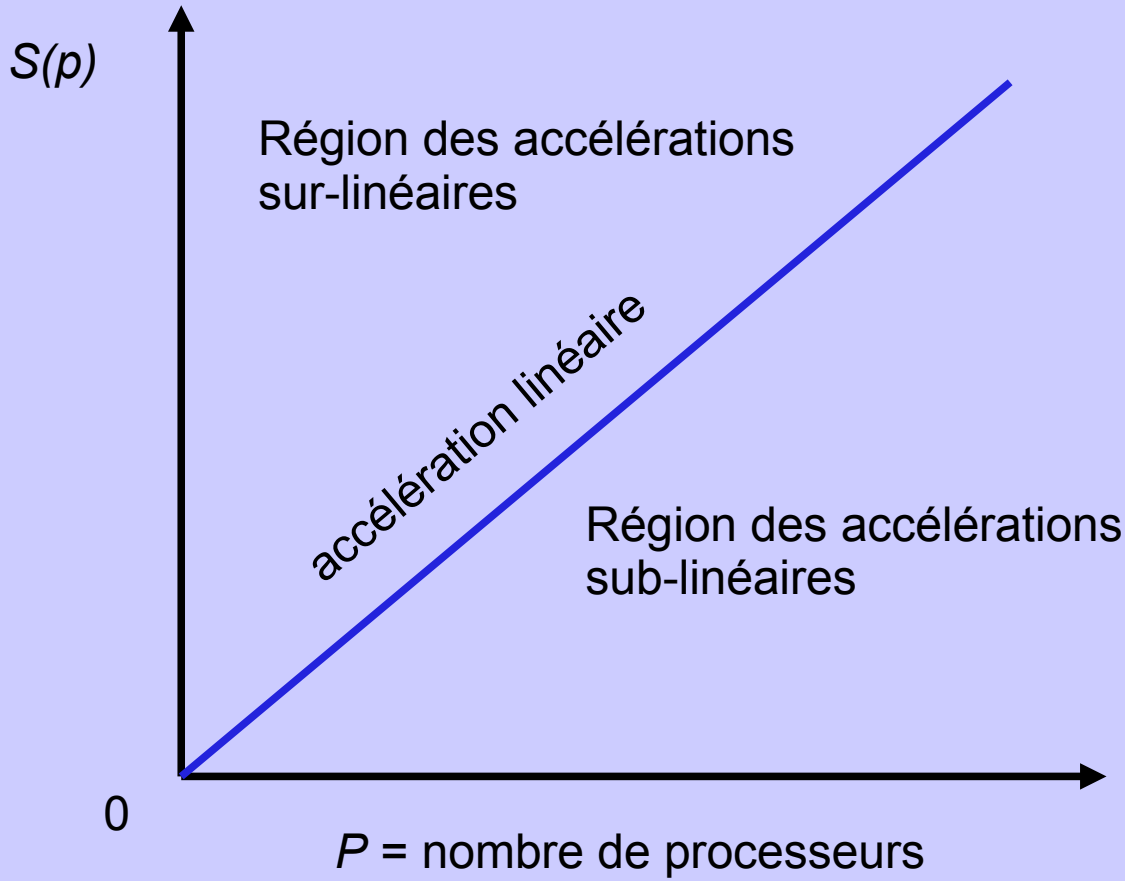
$$S(p) = T1 / Tp$$

pb : définition pas assez précise

Parallélisation : notion d'accélération

- Pour obtenir des résultats comparables il faut utiliser les mêmes définitions d'Ordinateur Séquentiel et de Programme Séquentiel
- Ordinateur Séquentiel
  - Ordinateur // configuré avec un seul processeur
  - Ordinateur séquentiel d'une puissance similaire à l'ordinateur //
- Programme Séquentiel
  - Programme // configuré pour s'exécuter sur un seul processeur
  - Programme séquentiel utilisant le même algo que le programme //
  - Programme séquentiel le plus rapide connu utilisant le même algo que le programme //
  - Programme séquentiel (ou le plus rapide) résolvant le même pb.
- Beaucoup de combinaisons, il faut préciser dans chaque cas

Parallélisation : notion d'accélération



Parallélisation : notion d'efficacité

- Soit ***T1(n)*** le temps nécessaire à l'algorithme pour résoudre une instance de problème de taille ***n*** avec un seul processeur, soit ***Tp(n)*** celui que la résolution prend avec ***p*** processeurs et soit ***s(n,p) = T1(n) / Tp(n)*** le facteur d'accélération.

  On appelle efficacité de l'algorithme le nombre :

$$E(n,p) = S(n,p) / p$$

- Efficacité = normalisation du facteur d'accélération

Parallélisation : Efficacité / Accélération

- exemple : Multiplication de matrices
  - Algorithme A
    - Temps en séquentiel : 10 minutes
    - Nombre de processeurs : 10
    - Temps en // : 2 minutes
    - Accélération : 10/2 = 5 (l'application va 5 fois plus vite)
    - Efficacité : 5/10 = 1/2
  - Algorithme B
    - Temps en séquentiel : 10 minutes
    - Nombre de processeurs : 3
    - Temps en // : 4 minutes
    - Accélération : 10/4 = 5/2 = 2,5 < 5
    - Efficacité : (5/2)/3 = 0,8 > 0,5

Parallélisation : La loi d'Amdahl

- Le temps d'exécution ***T1*** d'un programme séquentiel peut être décomposé en deux temps :
  - T_s consacré à l'exécution de la partie intrinsèquement séquentielle
  - $T_{//}$ consacré à l'exécution de la partie parallélisable

$$T1 = T_s + T_{//}$$

- Seul $T_{//}$ peut être diminué par la parallélisation

- Dans le cas idéal on obtiendra au mieux un temps $T_{//}/p$ pour la partie parallélisée

$$T_p \geq T_s + T_{//}/p$$

- L'accélération d'un programme sera donc limitée par le pourcentage de code intrinsèquement séquentiel qu'il contient.

Parallélisation : La loi d'Amdahl

- exemple en traitement de l'image : filtrage parallèle
  - Partie intrinsèquement séquentielle
    - capture
    - chargement
    - sauvegarde
  - Partie parallélisable
    - découpage
    - filtrage

Les Types d'ordinateurs parallèles

- Les machines vectorielles multi-processeurs :
  - Faible nombre de processeurs puissants (1 à 16)
  - Mémoire partagée
  - Limite atteinte, coût important
- Les multi-processeurs à mémoires distribuées :
  - Grand nombre de processeurs ordinaires à mémoire locale
  - Communication par envoi de messages à travers des réseaux de communication
  - Les processeurs ont leur propre séquenceur
- Les machines synchrones :
  - Très grand nombre d'éléments de calcul (4096 à 65536) de faible puissance avec une toute petite mémoire locale
  - Un séquenceur unique : exécution d'une même instruction sur des données différentes

Les Types d'ordinateurs parallèles

Tendance pour le parallélisme général

- On se dirige vers l'utilisation des multi-processeurs à mémoires distribuées :
  - Les machines à mémoire partagée ont atteint leur limite
  - Les machines synchrones sont trop rigides
  - Bon rapport coût/performance
- Futur : stations de travail reliées par un réseaux à très haut débit
  - développement d'OS spécifiques
  - développement de bibliothèques de communication (PVM, MPI)
  - Méta-Computing, Grille de calculs…

Parallélisation en TI

- le traitement d'images :
  - un des domaines où le parallélisme de traitement a été appliqué le plus tôt.
  - 1950's : propositions d'architectures massivement parallèles
  - 1970's : 1ère machines parallèles ILLIAC IV pour le TI.
  - nombreux systèmes matériels depuis les années 1980.

Parallélisation : systèmes dédiés

- systèmes matériels dédiés aux applications de TI
  se distinguent par divers critères :
  - domaine d'applications visé
  - contraintes :
    - performances
    - coût
    - complexité matérielle
    - consommation
    - encombrement
    - ...

Parallélisation : systèmes dédiés

- domaines :
  - systèmes industriels : robotique, contrôle qualité...
  - applications médicales : analyse d'échantillon, scanner...
  - véhicules intelligents
  - applications militaires et spatiales
  - télévision, cinéma
  - télésurveillance

Parallélisation : systèmes dédiés

- programmabilité des systèmes :
  - traitement systématique d'un flux d'images selon un algorithme prédéfini

    ↪ *Machine dédiée* : version optimisée de l'algorithme câblée
  - flexibilité dans l'exécution des commandes et programmabilité dans la mise en œuvre

    ↪ *Machine spécialisée* : optimisation générique des traitements les plus représentatifs de l'application.

Parallélisation : systèmes dédiés

- autres critères :
  - coût :

    ↪ large diffusion – application grand public
  - volume, consommation :

    ↪ application embarquée : véhicule, caméra intelligente...

Parallélisation : algorithmes de TI

Type de méthodes d'un point de vue de leur implication architecturale c-à-d des *mouvements de données* requis

- <u>traitements bas niveau</u> :
  - transforme une image par des opérations sur des combinaisons de pixels voisins *(filtrage)*

    ↪ mouvements de données réguliers, simples et systématiques *(quelles que soient les valeurs des pixels)*

    ⇒ *Machines dédiées* existantes en majorité

Parallélisation : algorithmes de TI

- traitements intermédiaires – analyse d'image :
  - extrait des caractéristiques d'une image *(régions, contours, motifs, mouvements, carte de profondeur,...)*

    ↪ pas d'homogénéité : divers données *(arbres, listes, graphes, matrices...)*, divers techniques *(statistique, déterministe...)* et des méthodes spécifiques *(vision, analyse numérique, théorie des graphes...)*

    ↪ mouvements souvent irréguliers et fortement dépendants des données.

  ⇒ peu de *machines dédiées* ⇒ ordinateurs parallèles généralistes

- traitements de haut niveau : idem *(héritent des représentations de l'analyse d'image)*

Les architectures

- *machine* = des opérateurs traitant des données
  opérateurs simultanément actifs ⇒ machine parallèle

- Classification de Flynn *(1972)* :
  contrôle des traitements *(flot d'instructions)* ⏩ flots de données

Flot d'instruction \ Flot de données	Unique	Multiple
Unique	**SISD** (von Neumann)	**SIMD** (tab de processeurs)
Multiple	**MISD** (pipeline)	**MIMD** (multi-processeurs)

Les architectures : *machine séquentielle*

SISD *(Single Instruction stream, Single Data stream)*

1 flot d'instructions ⏩ 1 seule donnée

- séquenceur unique
- processeur unique
- mémoire unique

Les architectures : *machines parallèles*

- machines parallèles : 3 classes
  - SIMD *(Single Instruction stream, Multiple Data stream)*

    1 unité de contrôle :

    plrs unités de traitement synchronisées ⏩ plrs flots de données
    - séquenceur unique
    - tableau de processeurs

  - MISD *(Multiple Instruction stream, Single Data stream)*

    1 unité séquence des traitements indépendants ⏩ 1 seul flot de données
    - pipeline d'opérateurs

Les architectures : *machines parallèles*

- MIMD *(Multiple Instruction stream, Multiple Data stream)*

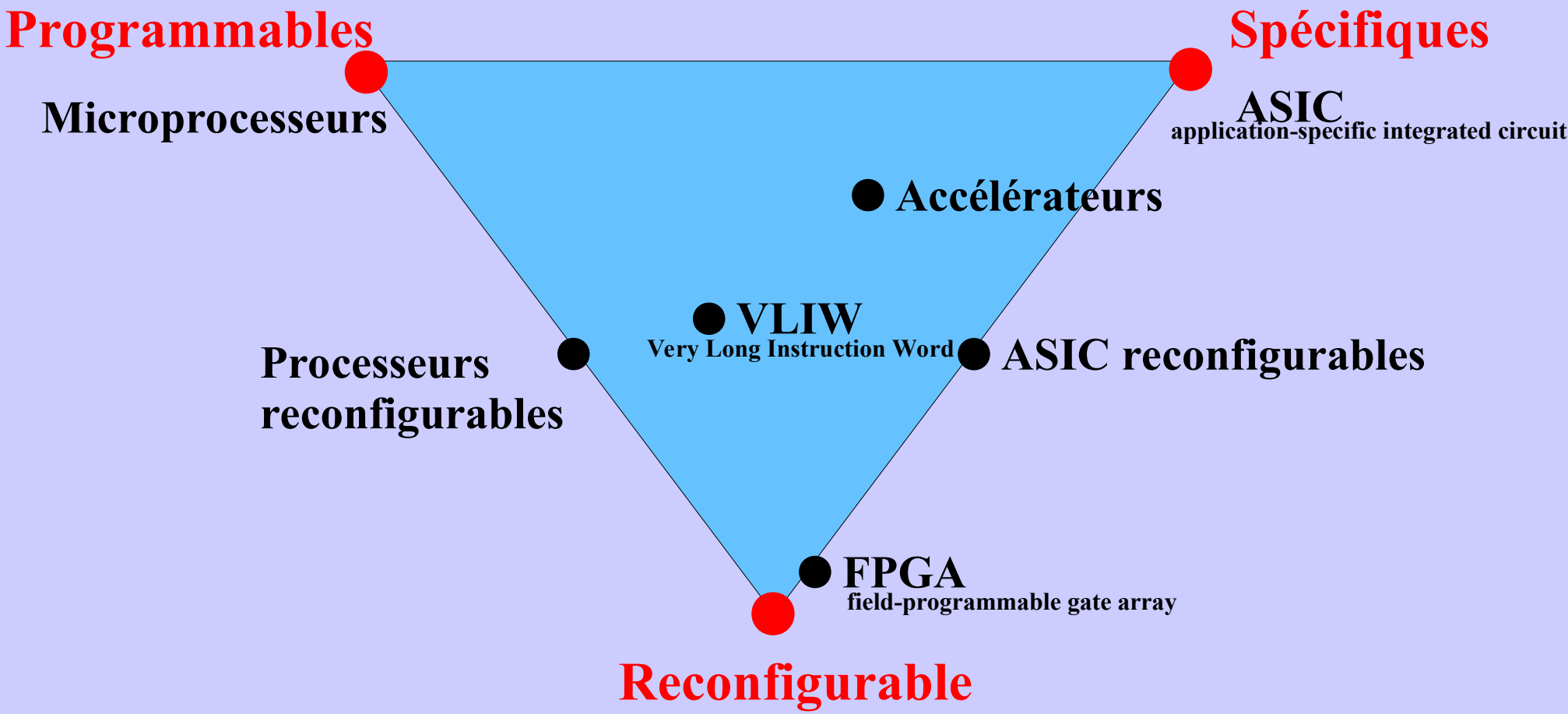
  plrs unités de contrôle *(ordinateurs indépendants)*

  ⏩ chacun son flot de données

  - processeurs autonomes
  - mémoire partagée ou distribuée

- SIMD et *pipeline* plus courants et meilleur rapport performance/ coût pour le TI

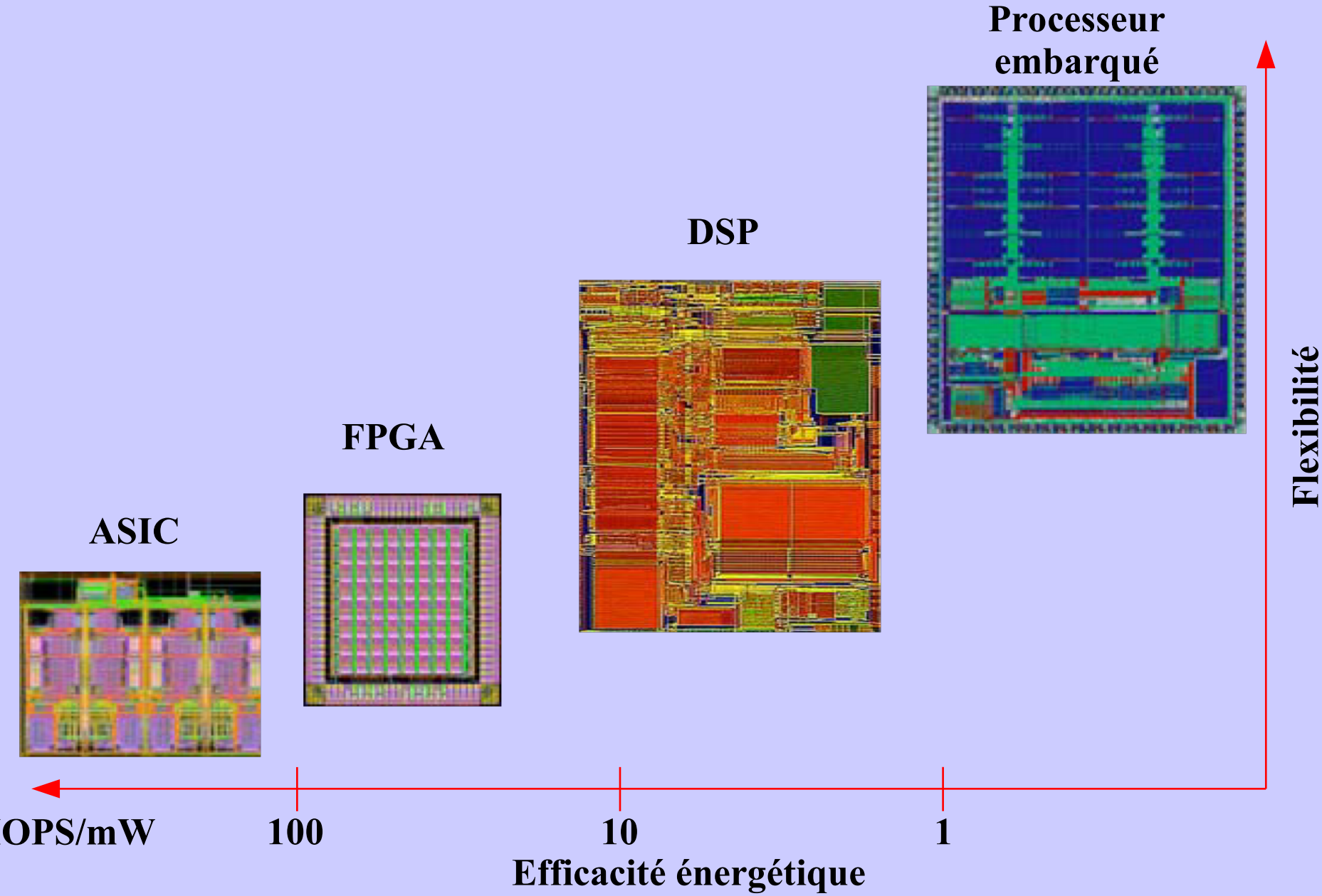
Technologie : *3 grandes classes*



Technologie : *3 grandes classes*

- composants standards – microprocesseurs généralistes, DSP :
  - excellente programmabilité, outils logiciels, optimisation des performances, technologie performante, faible coût.
- composants dédiés, spécifiques : circuits intégrés VLSI, ASIC :
  - optimal pour une application donnée
    mais coût de développement élevé et rapidement obsolète.
- circuits reconfigurables, FPGA :
  - à la fois efficace et programmable mais peu pratique

Technologie : *Compromis flexibilité énergie*



Technologie : *3 grandes classes*

	Approche Programmée		Approche Spécifique	Approche Reconfigurable	
	CPU	DSP	ASIC	FPL	RC
Puissance de traitement	+	++	++++	++ +?	+++
Faible Consommation	-	+	++++	?	++
Flexibilité	++++	++	----	+++	++

Field Programmable Logic

Reconfigurable Computing

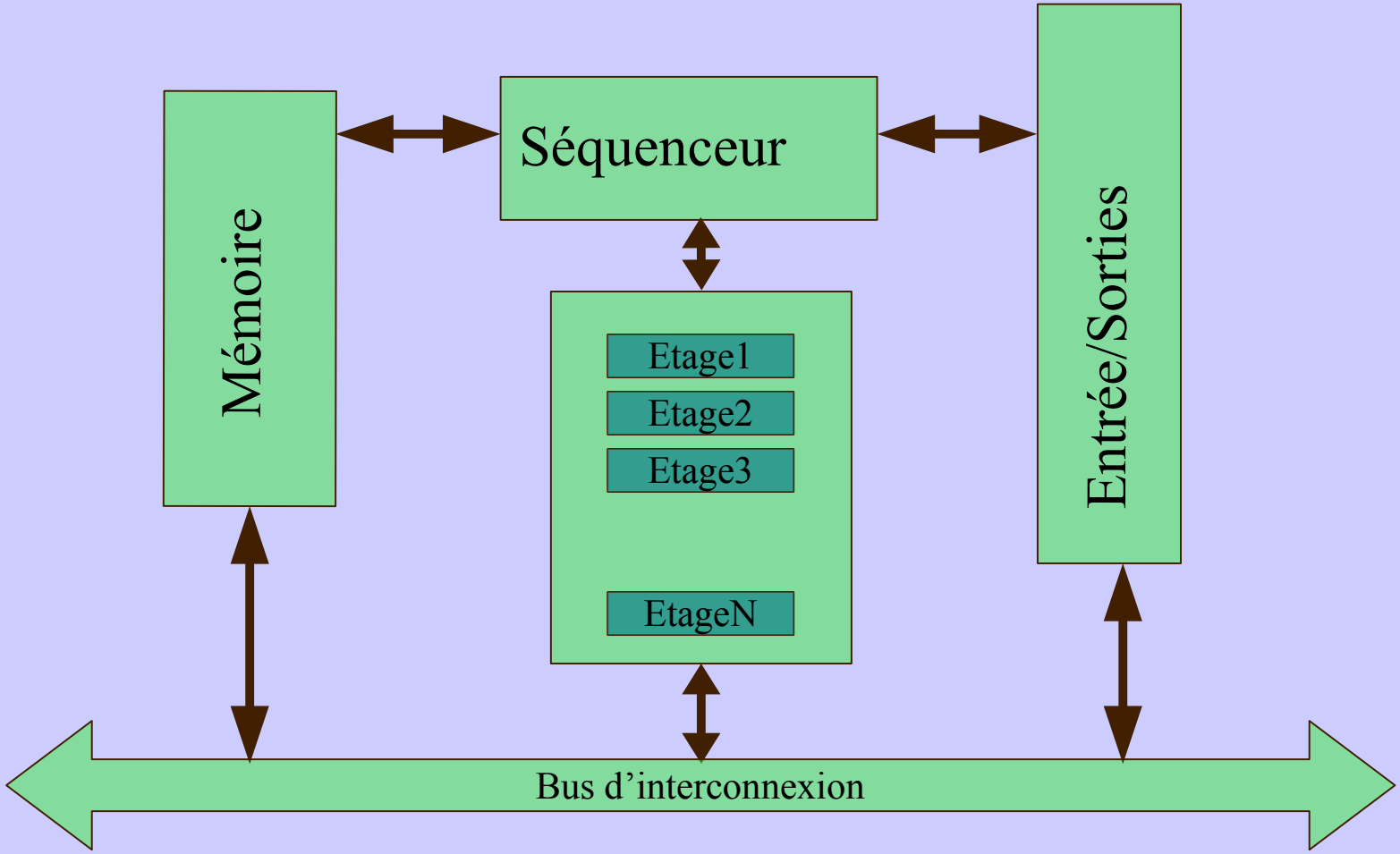
Technologie : *2 familles de réalisations*

- parallélisme de flux :
  - traitement à la volée, au sortir de la caméra
  - exécution en *pipeline* d'opérations successives
  - traitements bas niveau
  - réalisation simple, peu programmable mais à faible coût
- parallélisme de données :
  - données stockées en mémoire, manipulées sur place
  - parallélisme important → 1 unité de traitement / pixel
  - traitements bas niveau 1 des aspects d'analyse d'images
  - bonne programmation, coût très élevé

Parallélisme de flux : le *pipeline*

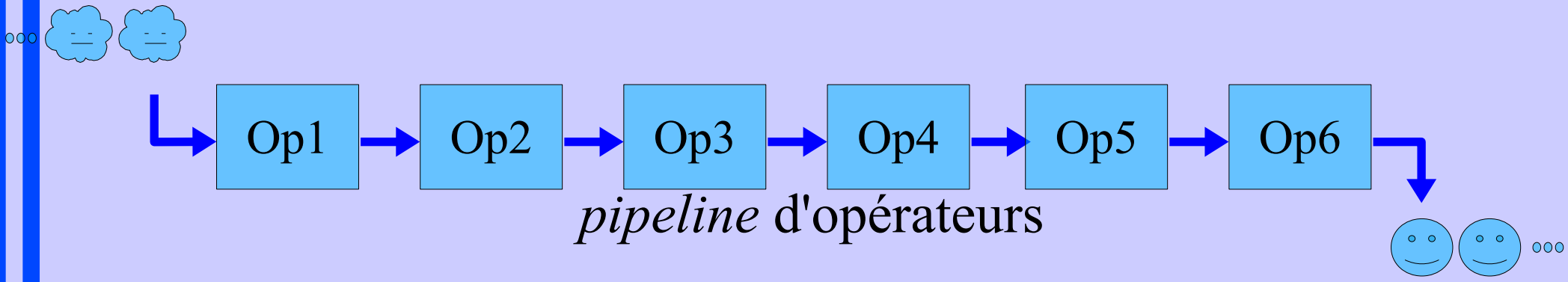
- Décomposition d'un algorithme en une succession de ***T*** tâches qui s'enchaînent les unes aux autres
- Subdivision de l'unité de calcul en ***N*** étages
- Il faut remplir le *pipeline* (***N*** cycles)
- L'accélération obtenue est $$A = \frac{N * T}{N + T - 1}$$
- L'accélération maximale correspond au nombre ***N*** d'étages (théorique)
- Rapport performance/coût intéressant

Parallélisme de flux : le *pipeline*



Parallélisme de flux : le *pipeline*

- image balayée de manière régulière – par exemple par ligne
  - pixels → opérateurs dédiés → données *(pixels)*



pipeline d'opérateurs

- temps de traitement :

$$(P + D) * \tau$$

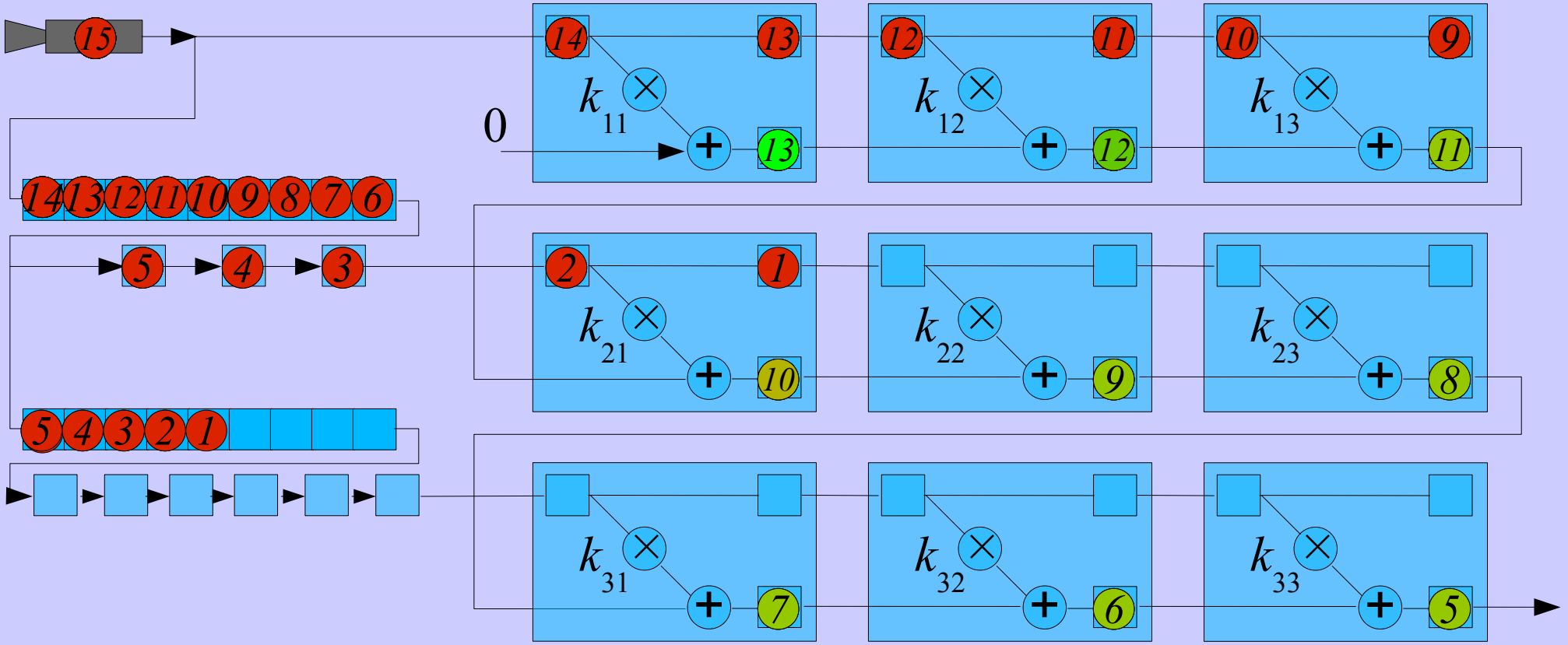
τ : tps 1 traitement 1 pixel, P : nb opérateurs, D : nb pixels

Parallélisme de flux : *pipeline* systolique

- Généralisation de la structure *pipeline*
- ensemble de cellules interconnectées pouvant réaliser une opération élémentaire
- Les informations transitent entre les cellules selon la technique *pipeline*, mouvements réguliers et synchrones
- Avantages :
  - modularité et facilité d'intégration
  - accélération importante pour du calcul de type matriciel
    ↪ traitement d'images de bas niveau
- Rapport performance/coût intéressant très élevé

Parallélisme de flux : *pipeline* systolique

- exemple : convolueur *pipeline* systolique
  - Pb : utilise le pixel d'origine et son voisinage
    - ↪ balayage par les lignes
    - ↪ mémorise les lignes précédentes dans des registres à décalage



Parallélisme de flux : *pipeline*

- exemple : histogramme et transformée de Hough
  - histogramme :
    - flot de pixels *(x,y,v)* *v niveau de gris du pixel (x,y)*
      → *incrémenter le tableau à l'index v*
  - transformée de Hough :
    - pour chaque pixel détecté (x,y)
      → *incrémente la case mémoire solution de*

$$\rho = x \cos\theta + y \sin\theta$$

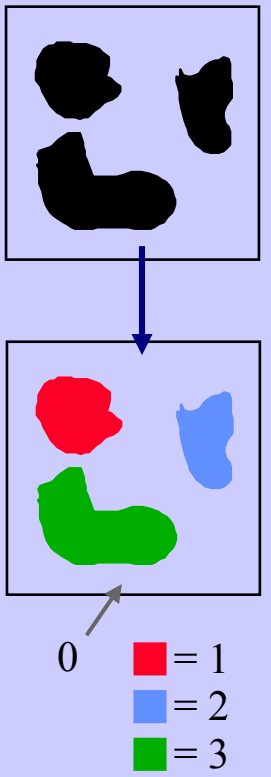
  - associe un triplet *(x,y,v)* à une adresse mémoire

Parallélisme de flux : *pipeline*

- exemple : étiquetage de composantes connexes

  Algorithme de Rosenfeld :

  - parcours l'image ligne par ligne
  - pour chaque pixel *(on connaît les voisins précédents)*
    - si aucun voisin étiqueté : nouvelle étiquette
    - si tous ses voisins même étiquette : affecte celle-ci
    - si ses voisins étiquetés ≠ : relie les composantes choisi étiquette et *màj* table de correspondance
  - parcours de l'image pour *màj* les étiquettes selon la table de correspondance



Parallélisme de flux : *pipeline*

- exemple : *Morpho-math* dilatation et érosion
  - formule d'Haralick, version *1D* (1987) :

$$(f \oplus k)(x) = \max_{\substack{y \in K \\ x-y \in F}} \{f(x-y) + k(y)\}$$

$$(f \to k)(x) = \min_{y \in K} \{f(x+y) - k(y)\}$$

x : coordonnée de l'image

y : coordonnée de l'élément structurant

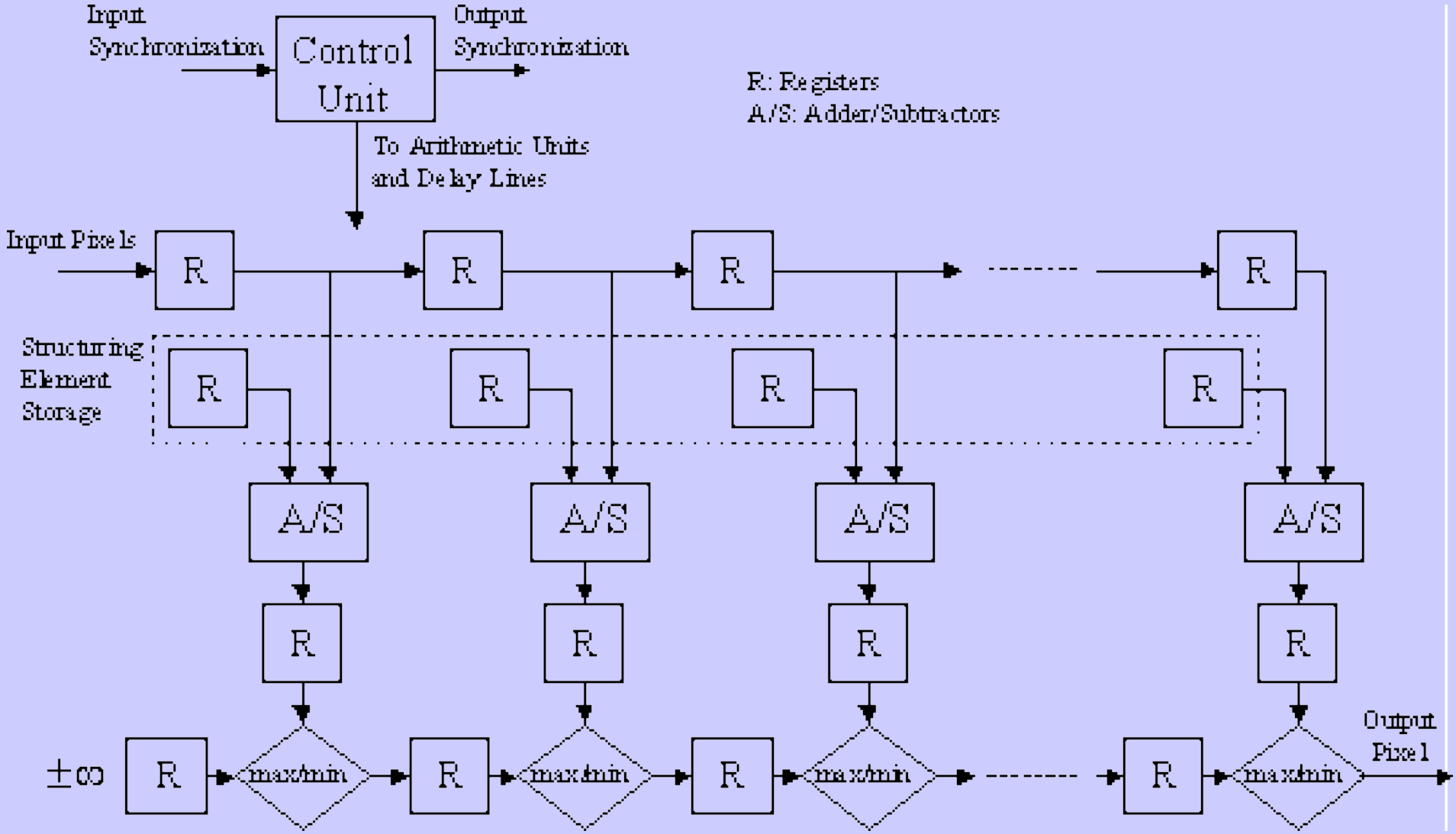
f : image en niveaux de gris

k : élément structurant en niveaux de gris

F, K : domaine de niveaux de gris de l'image et de l'élément structurant

Parallélisme de flux : *pipeline*

- exemple : *Morpho-math* dilatation et érosion



Parallélisme : *pipeline* flot de données

- Suppression du compteur ordinal
- Séquencement donné par les instructions elles-mêmes
- Structure de données à plusieurs champs notée *jeton*
  - la donnée propre : un champ
  - contrôle du flot d'instructions : les autres champs
- Instruction exécutée que lorsque tous ces opérandes disponibles
- Les programmes : graphes orientés
  - les nœuds sont les actions
  - les arcs les chemins empruntés

Étude d'un processeur *pipeline* à flot de données

- Calcul d'un gradient
  - pour un pixel donné : lire les valeurs des 8 voisins, calculer nouvelle valeur, écrire cette valeur
  - la parallélisation concerne la
    - parallélisation du traitement
      - préparation des adresses des 8 pixels voisins
      - préparation de l'adresse du pixel à écrire
    - parallélisation des données
      - le calcul de chaque pixel est indépendant des autres
      - chaque processeur traitera N/p lignes x N pixels

Étude d'un processeur *pipeline* à flot de données

	1 processeur	8 processeurs	16 processeurs	32 processeurs
Soustraction Constante	0,696 s	87 ms	43,5 ms	21,75 ms
Sobel	4,81 s	601,25 ms	300,62 ms	150,31 ms
Histogramme	0,749 s	94,42 ms	47,6 ms	24,2 ms

Temps d'exécution pour une image 512x512

Étude d'un processeur *pipeline* à flot de données

- le temps de traitement peut être optimisé
  - par exemple pour le calcul du gradient
    - fenêtre glissante qui minimise les accès mémoires
  - Il faut tenir compte des spécificités architecturales ou logicielles
- implémentation d'une application de T.I :

  → analyse des problèmes de synchronisation entre les différents tâches

  ⇒ graphe de flot de données