École thématique ARCHI'05

Une méthode de conception de systèmes sur puce

(de l'intégration d'applications)

Frédéric PÉTROT

Laboratoire TIMA
Institut National Polytechnique de Grenoble

Plan de la présentation

- Généralités
  - Systèmes sur puce vs Systèmes informatiques
  - Architectures cibles
- Une méthode d'intégration d'applications
  - Principes
  - Problèmes
  - Flot de conception et illustration

Plan de la présentation

- Généralités
  - Systèmes sur puce vs Systèmes informatiques
  - Architectures cibles
- Une méthode d'intégration d'applications
  - Principes
  - Problèmes
  - Flot de conception et illustration

Systèmes sur puce vs systèmes informatiques

Ressources limitées (coût, consommation) :

- taille mémoire faible
- petite taille de mots
- fréquence de fonctionnement (relativement) faible

Se focaliser sur l'efficacité :

- programmation à bas niveau (asm/C)
- systèmes d'exploitation minimalistes
- architectures spécialisées
  - coprocesseurs *ad-hoc*
  - processeurs spécialisés : DSP, NP, GPU
  - communications spécialisées

Systèmes sur puce vs systèmes informatiques

Application(s) $\Rightarrow$ donnée du problème
Environnement $\Rightarrow$ contraintes

- problème d'optimisation multi-critères
- coût non formulable mathématiquement
- réalisation pouvant faire usage de matériel *ad-hoc*

Spécialisation dans des domaines d'application

- application à haut débit de données
- automates de contrôle

$\Rightarrow$ langages spécifiques à un domaine (PN, ET, SL, ...)
$\Rightarrow$ contraintes spécifiques à un domaine

Systèmes sur puce vs systèmes informatiques

Systèmes sur puce vs systèmes informatiques

Plan de la présentation

- Généralités
  - Systèmes sur puce vs Systèmes informatiques
  - Architectures cibles
- Une méthode d'intégration d'applications
  - Principes
  - Problèmes
  - Flot de conception et illustration

Architectures système

Architectures cibles

architecture	matériel	logiciel
SIMD	++	−−
VLIW	−−	++
MIMD	?	?

Cibles de l'implantation :

- ordinateur + carte FPGA
- carte spécialisée DSP + coprocesseurs *ad-hoc*
- système uni ou multiprocesseur intégré sur puce

Architectures système : ILP vs TLP

Instruction Level Parallelism : performance séquentielle

- super-scalaire
- simultaneous multithreading (SMT $\Rightarrow n$ tâches)

Complexité effarante !

- bus très larges
- prédiction de branchement avancée
- renommage de registres
- exécution désordonnée
- caches non bloquants

En 250 ps, ... (temps e^- met pour traverser 5 cm de CU)
$\Rightarrow$ coût + consommation inacceptable pour CE

Architectures système : ILP vs TLP

Task Level Parallelism : performance parallèle

- processeurs plus « simples »
- utilisation de coprocesseurs naturelle
- bien adapté à certaines applications

Mais :

- révolution culturelle pour programmer
- compilateurs (?)
- recouvrement calcul/communication explicite
- non adapté à certaines applications

$\Rightarrow$ Parallélisme à gros grain semble inévitable

Tendance CMP : multiprocesseurs $\pm$ généralistes

Gaming : Cell PS/3 (Sony/IBM/Toshiba)

- 8 processeurs spécialisés (APU), sans caches
- 1 PowerPC G4 SMT (2 threads)
- 234 millions de transistors, $\approx$ 4GHz

Web Servers : Niagara (Sun)

- 8 Ultra-sparc like
- 4 *threads* d'exécution matériels par processeur
- 1.2 GHz

General purpose : BCM1480 (Broadcom)

- 4 Mips64
- 800-1200 MHz
- 64-bit PCI-X, Gigabit Ethernet MACs (GMII), SPI-4/HT

Tendance CMP : multiprocesseurs $\pm$ généralistes

Intérêts :

- latence bien plus faible qu'en multi-chip
- bande passante interne quasi-gratuite

Problèmes :

- coût pour le *consumer* (surface + rendement)
- consommation
- programmation, ...

Plan de la présentation

- Généralités
  - Systèmes sur puce vs Systèmes informatiques
  - Architectures cibles
- Une méthode d'intégration d'applications
  - Principes
  - Problèmes
  - Flot de conception et illustration

Problématique

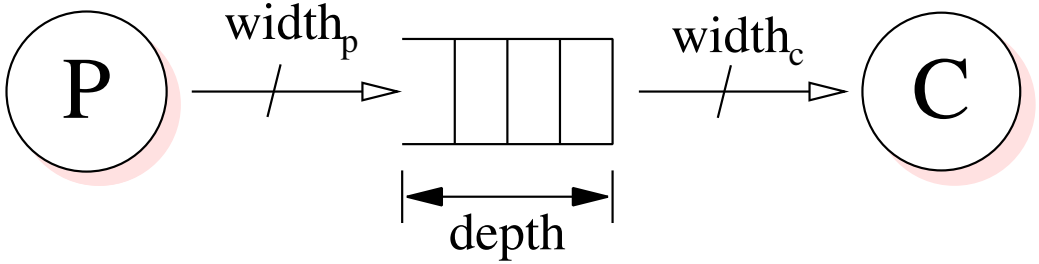
Conception sur plate-forme :

- extension d'un existant :
  - processeurs/mémoires/interconnexions connus
  - composants de communication/pilotes
  - techniques de conception/validation éprouvées
- technologie cible identique
- généralement mêmes horloges

Triplet :(système, nouvelle application, contraintes)

Principes : modèle des *réseaux de Kahn*

Réseaux de Kahn : adaptés aux flux de données



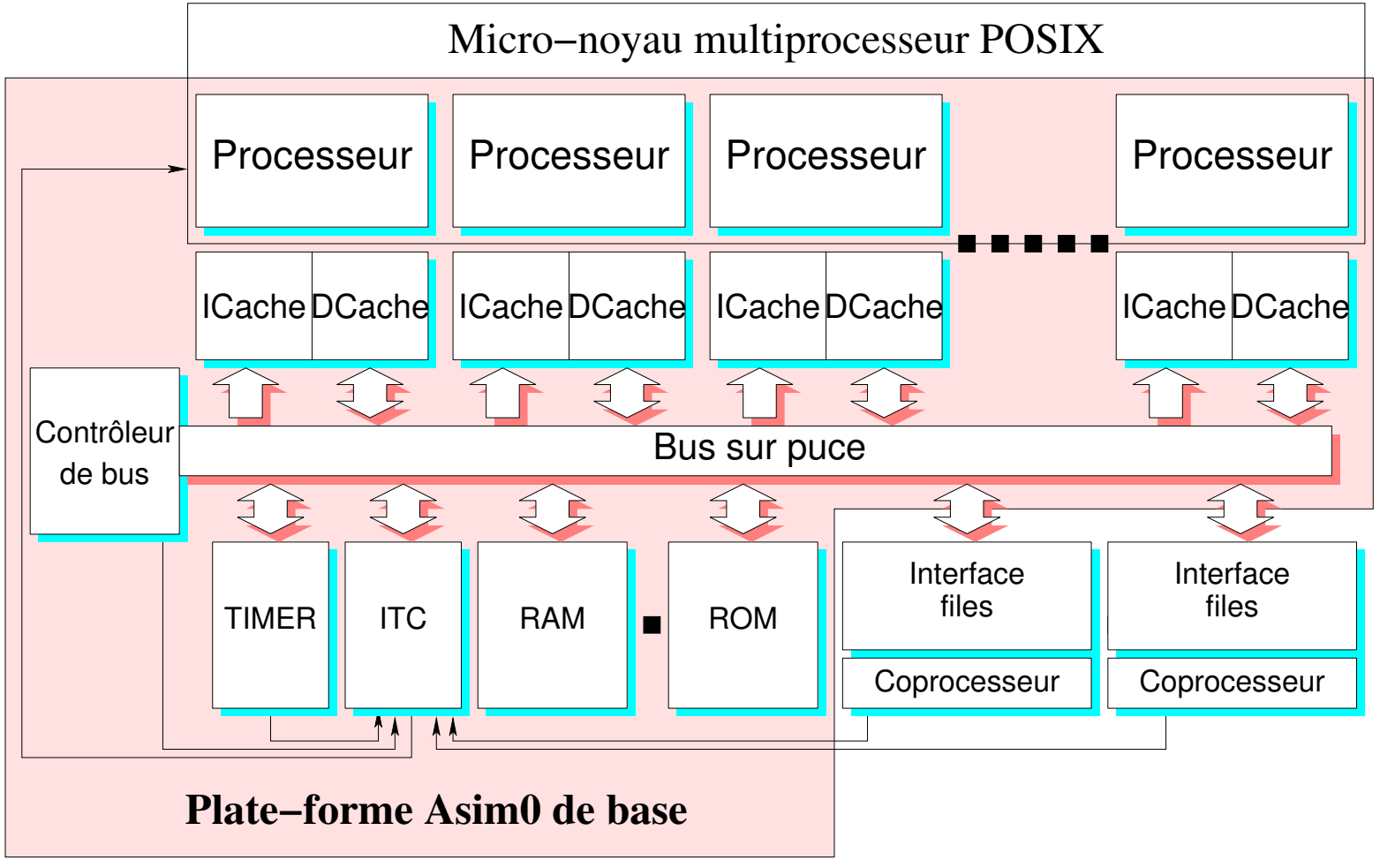
Propriétés théoriques :

- pas de pertes de donnée
- ordre d'émission = ordre de réception
- fonction indépendante des dates/durées d'exécution

Extensions pratiques :

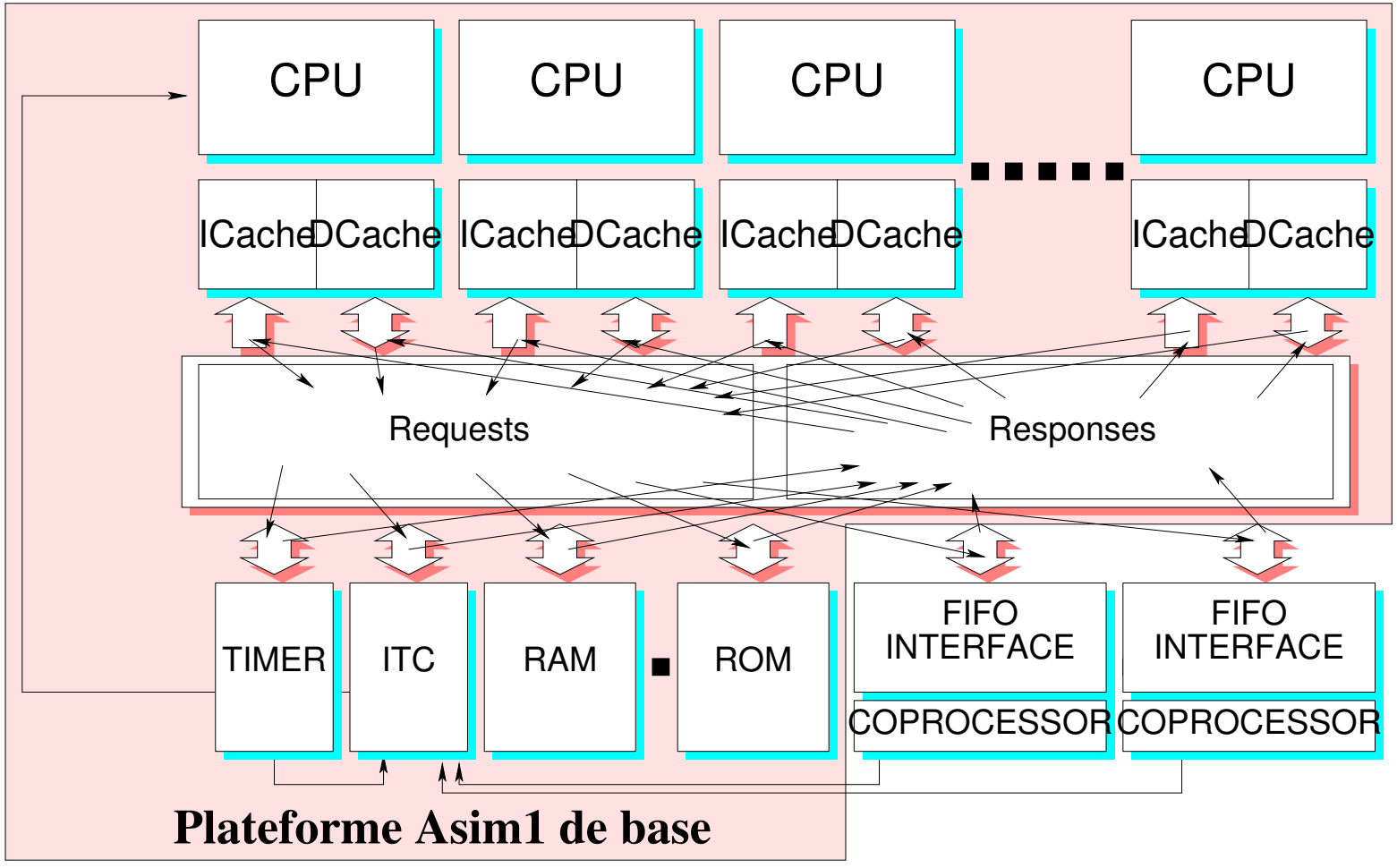
- FIFO bornées
- FIFO potentiellement bi-synchrones en matériel

Principes : architecture système *asim0*



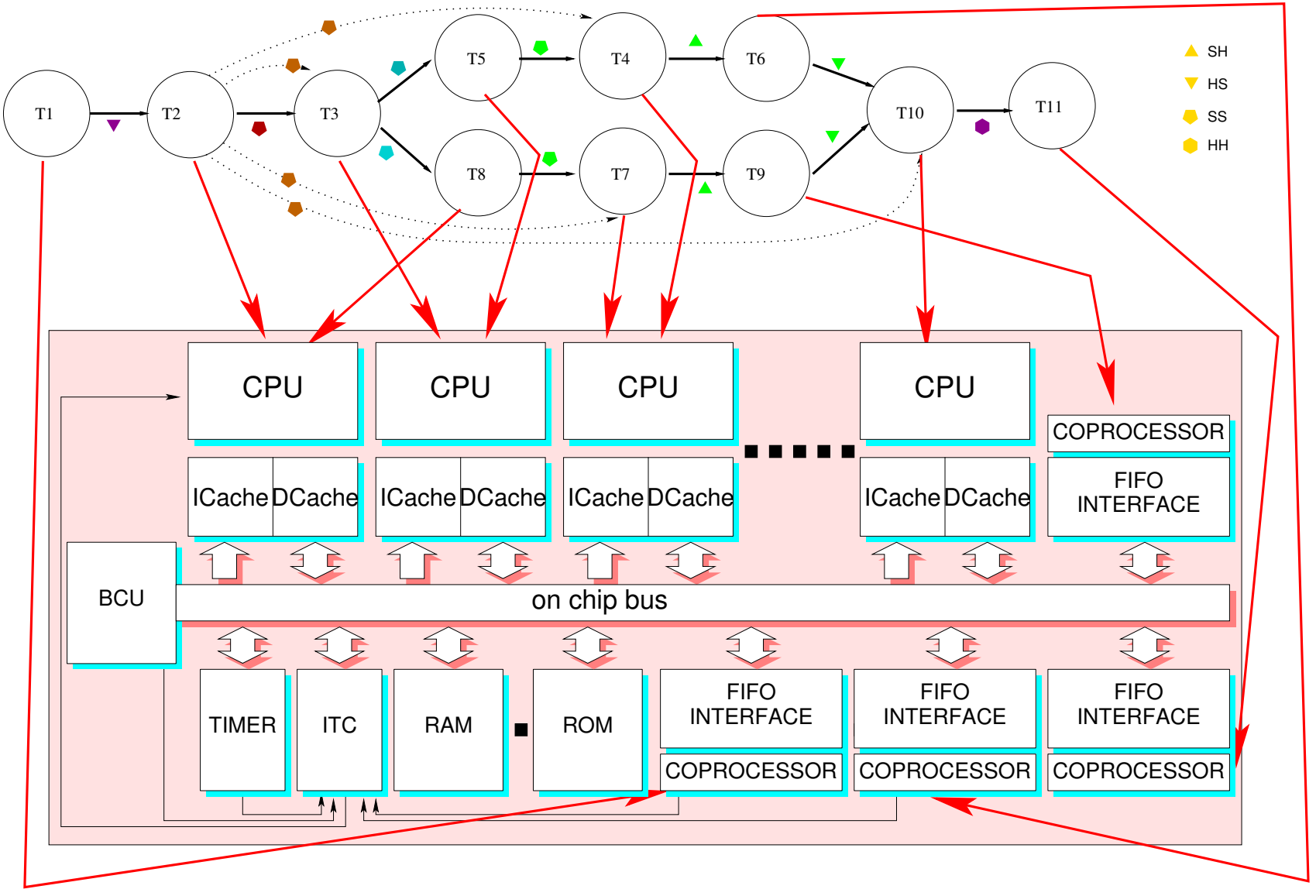
Simulable au niveau cycle

Principes : architecture système *asim1*



Simulable au niveau cycle

Principes : assignation des tâches



Principes : analyse

Restrictions : communications *système* limitées aux files
Mais :

- nature de la communication indépendante de la nature des tâches
- implantation de la communication très variées
  - par le processeur à l'aide d'interruptions
  - par le processeur en attente active
  - par un DMA
  - ...
- passage du logiciel au matériel cohérent par construction
- limite l'imagination du concepteur à des choses connues

Principes : analyse

Restrictions : architecture limitée à des instances d'**asim0/asim1**
Mais :
concevoir une plate-forme $\neq$ implanter une application sur une plate-forme

- conception des composants
- définition du support de communication
- choix de modèles de communication abstraits
- développement/choix d'un noyau
- ...

Réalité industrielle actuelle :
Semiconductor houses / system houses / software houses

Principes : migration logiciel → matériel

Migration exploratoire :

- co-simulation tâche (hôte)/plate-forme **asim0**(cycle)
- exploration d'architecture aisée

Migration réelle :

- réécriture de la tâche pour la synthèse
- vérification de la fonction sur l'hôte
- synthèse
- exécution par simulateur cycle

Dans les 2 cas :

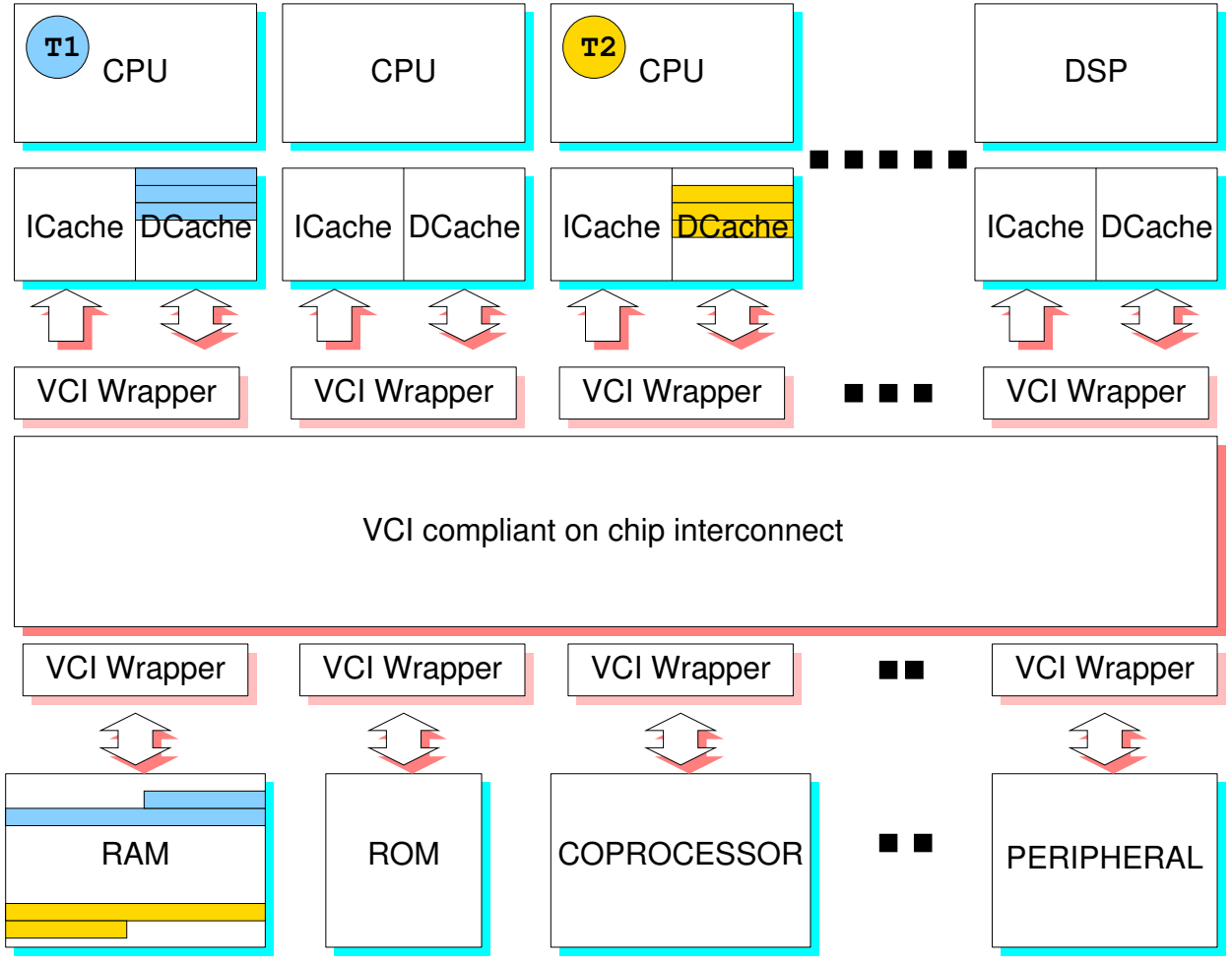
- changer la nature de la file
- création d'une plate-forme étendue

Plan de la présentation

- Généralités
  - Systèmes sur puce vs Systèmes informatiques
  - Architectures cibles
- Une méthode d'intégration d'applications
  - Principes
  - Problèmes
  - Flot de conception et illustration

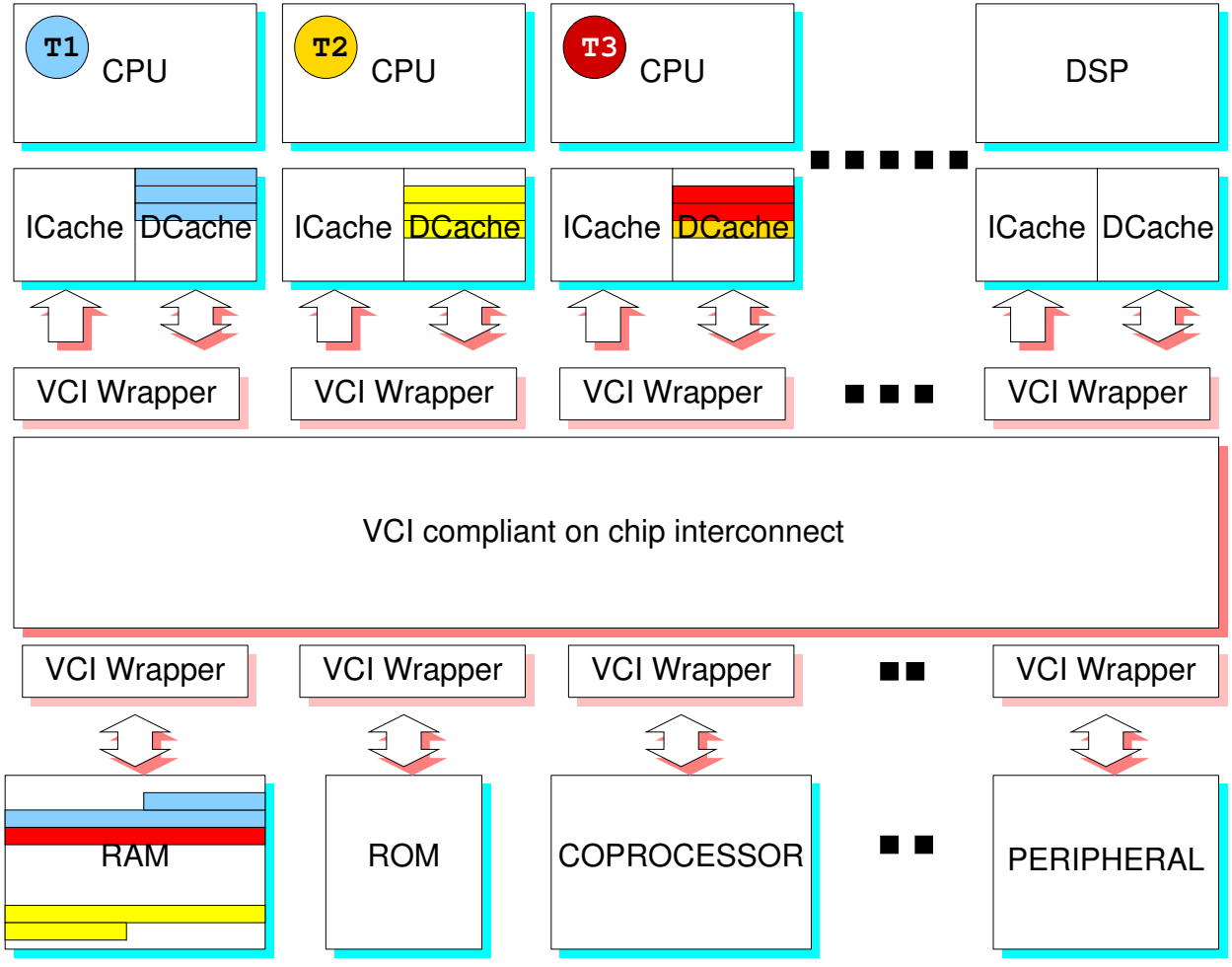
Problème : Cohérence des caches

Migration de tâches :



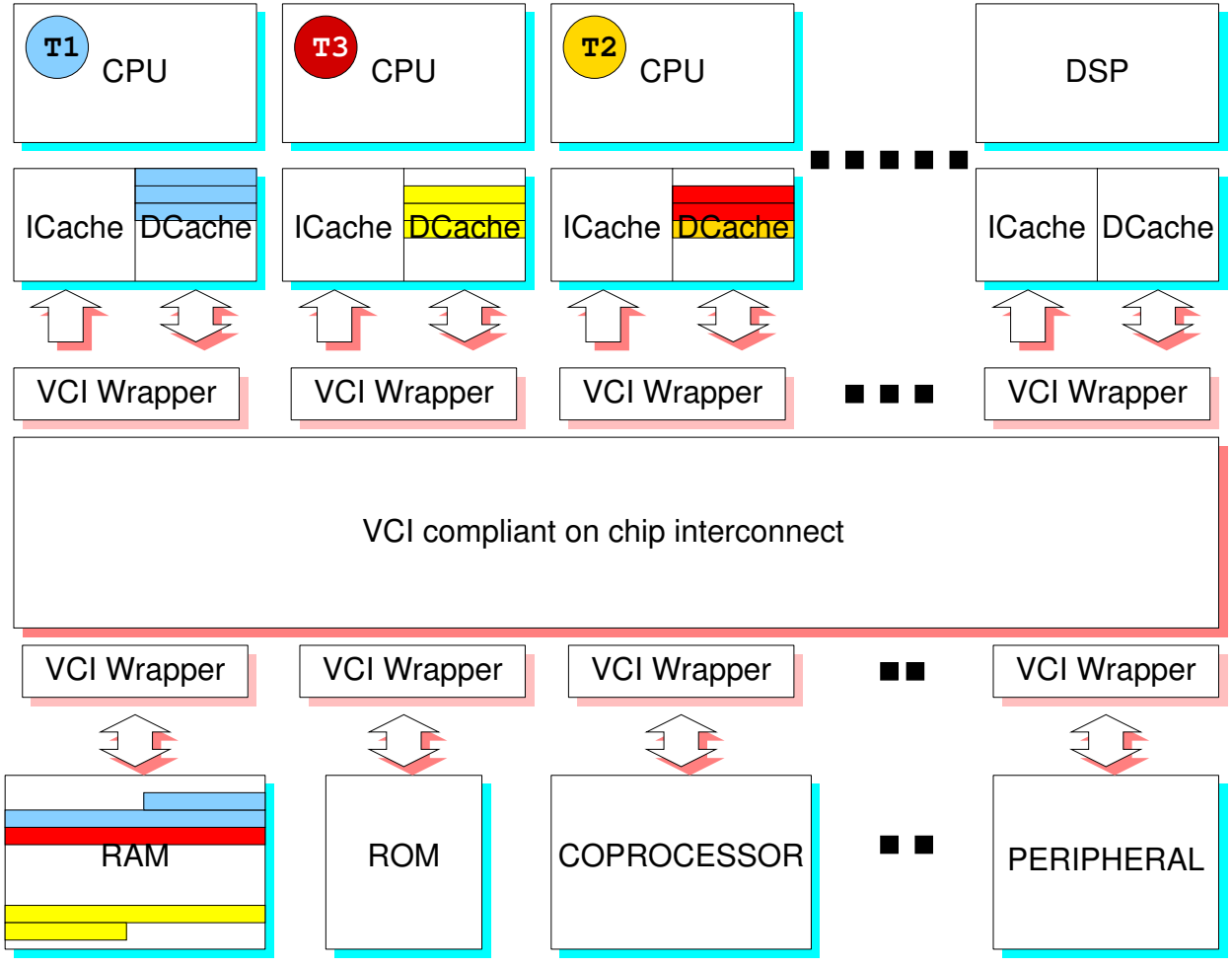
Problème : Cohérence des caches

Migration de tâches :



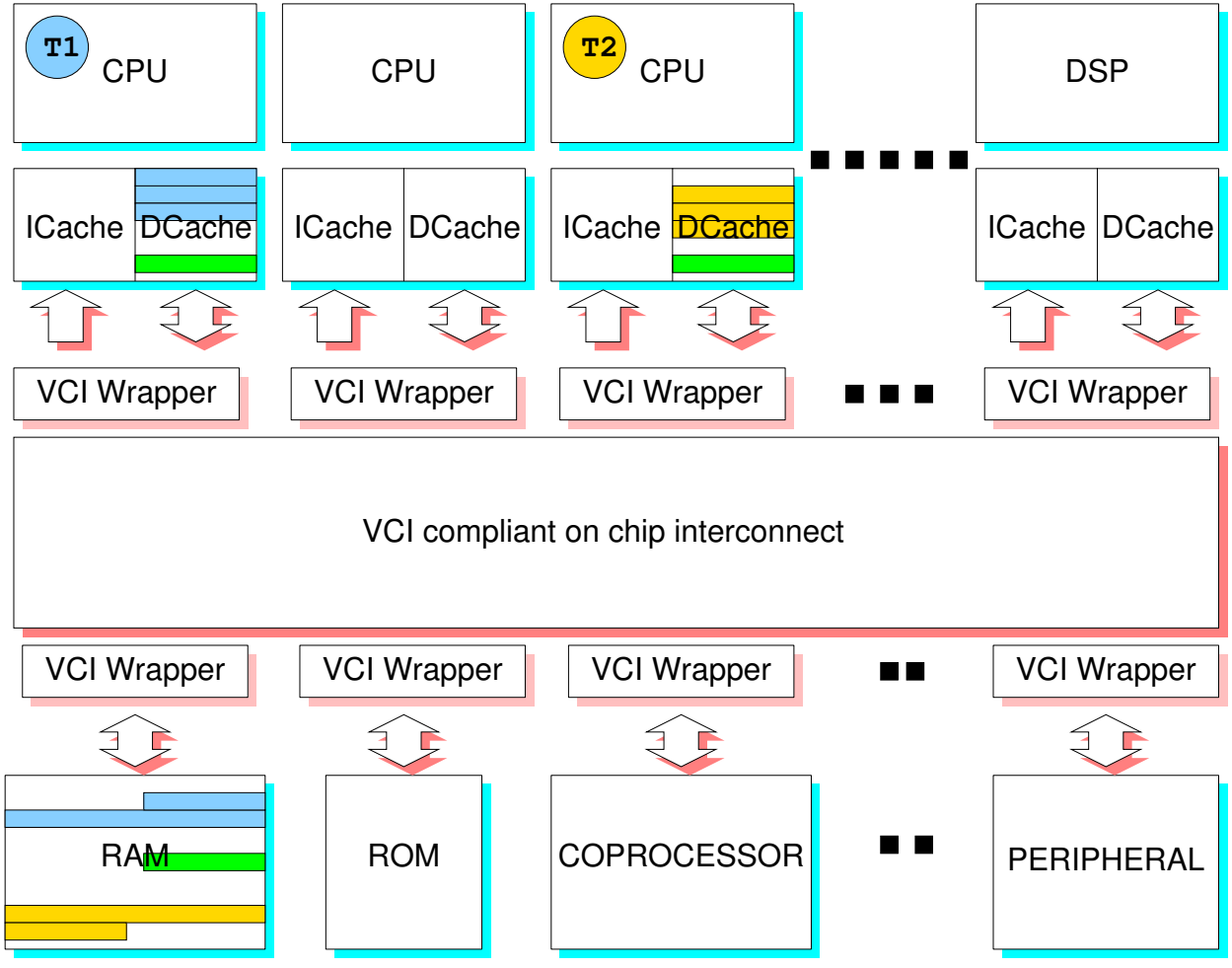
Problème : Cohérence des caches

Migration de tâches :



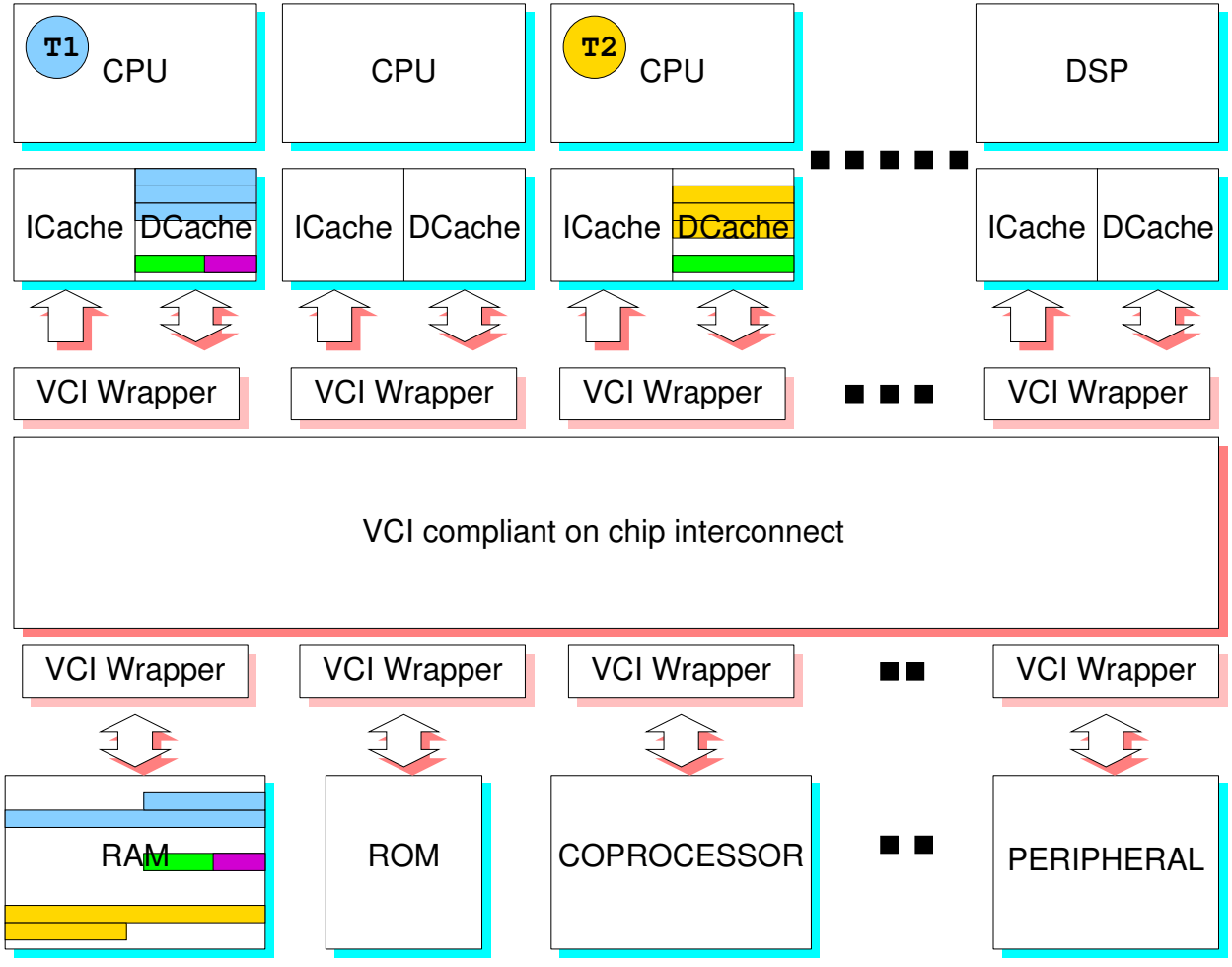
Problème : Cohérence des caches

Cohérence des données partagées :



Problème : Cohérence des caches

Cohérence des données partagées :



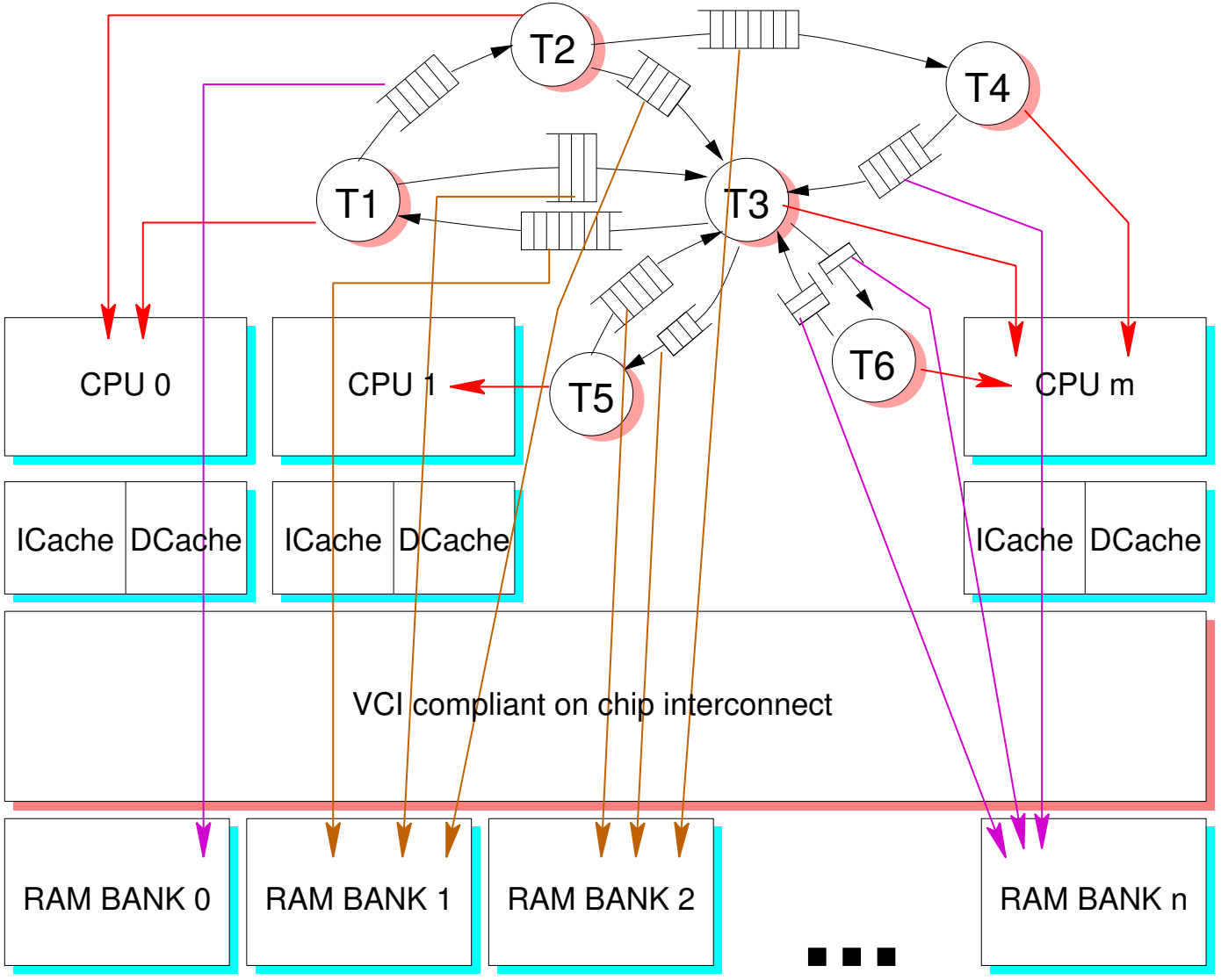
Cohérence : solution logicielle

Solution simple :

- pas de migration de tâches !
  - un noyau par processeur
  - tâche ne s'exécute que sur « son » processeur
- données locales à un processeur cachées ;
- données partagées entre processeurs non-cachées ;

Cohérence : solution logicielle

Solution simple :



Cohérence : solution logicielle

Désignation d'un processeur :

```
pthread_attr_t attr =  .procid = 3;
...
pthread_create(&tid, &attr, func, arg);
```

Désignation d'une mémoire :

- P segments locaux pour P processeurs ;
- N segments partagés ;
- utilisation dans une *libc* étendue.

```
void *local_malloc(unsigned int n);
void *_local_malloc(unsigned int lseg, unsigned int n);
void *shared_malloc(unsigned int sseg, unsigned int n);
```

Cohérence : solution matérielle

Directory based :

- complexité en $O(\frac{m}{b}\, p^2)$
- implantation matérielle et protocole
  - invalidation (moins de trafic et de trashing, plus de latence)
  - mémoire devient aussi un maître
  - caches deviennent aussi des esclaves
  - très complexe en *write-back* (et *write-allocate*)
  - assez simple en *write-through*

Cohérence : analyse quantitative

Hypothèses :

- les *miss* instructions sont négligés ;
- chaque processeur exécute une instruction RISC par cycle ;
- la latence moyenne sur un NoC VCI 32 bits est $\approx$ 40 cycles ;
- la politique d'écriture est *write-through* avec tamponnage ;
- il y a 20% de lectures, et 10% de ces lectures sont partagées ;
- le taux de *miss* moyen est de 15%, et le *hit* prend 1 cycle.

Cohérence : analyse quantitative

Cycles par instruction sans caches de données

$$\text{CPI} = (0.8 \times 1 + 0.2 \times 41) = 9$$

Cycles par instruction avec la technique logicielle

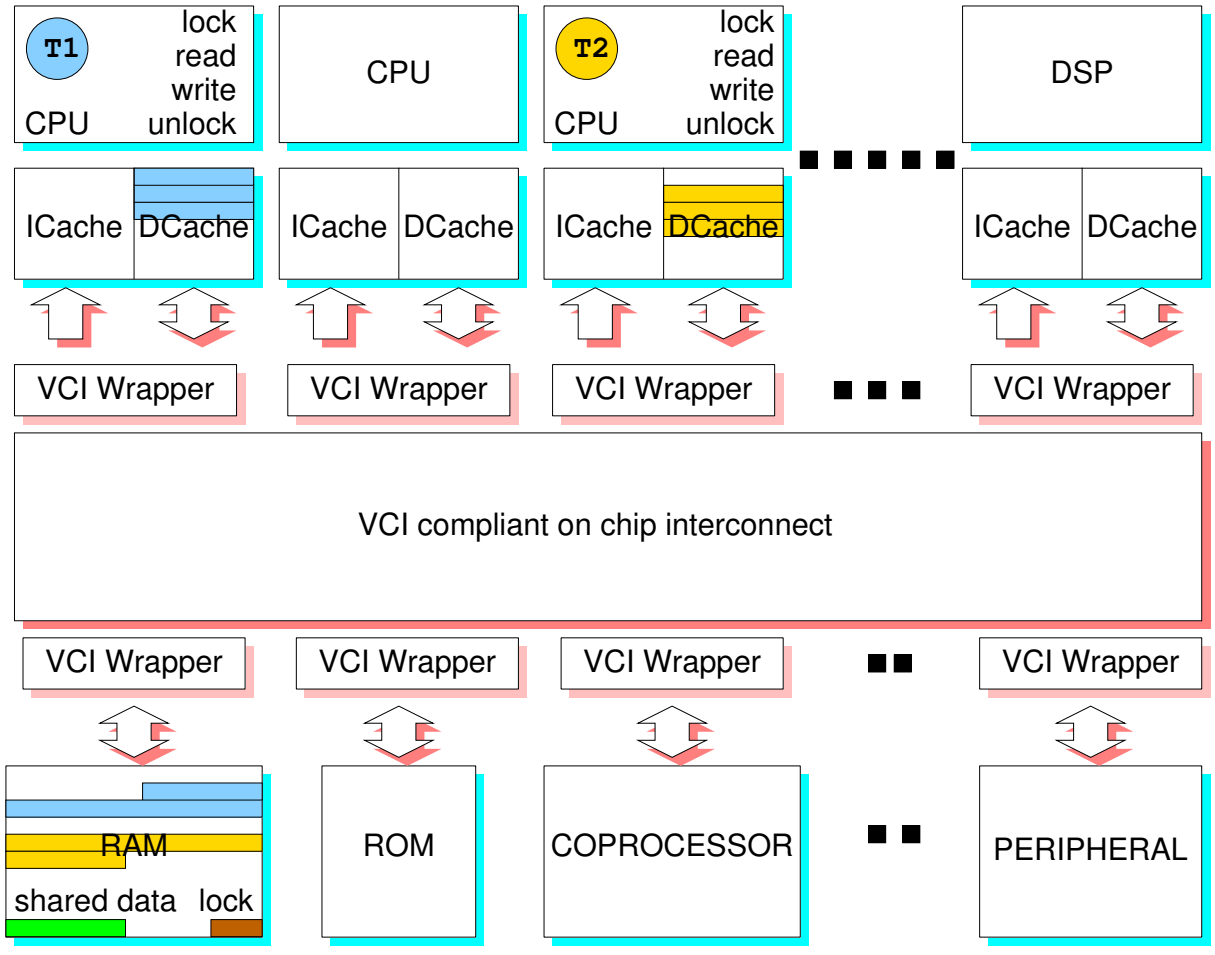
$$\text{CPI} = (0.8 + 0.2 \times (0.1 \times 41 + 0.9 \times (0.85 \times 1 + 0.15 \times 41))) = 2.88$$

- matériel inchangé

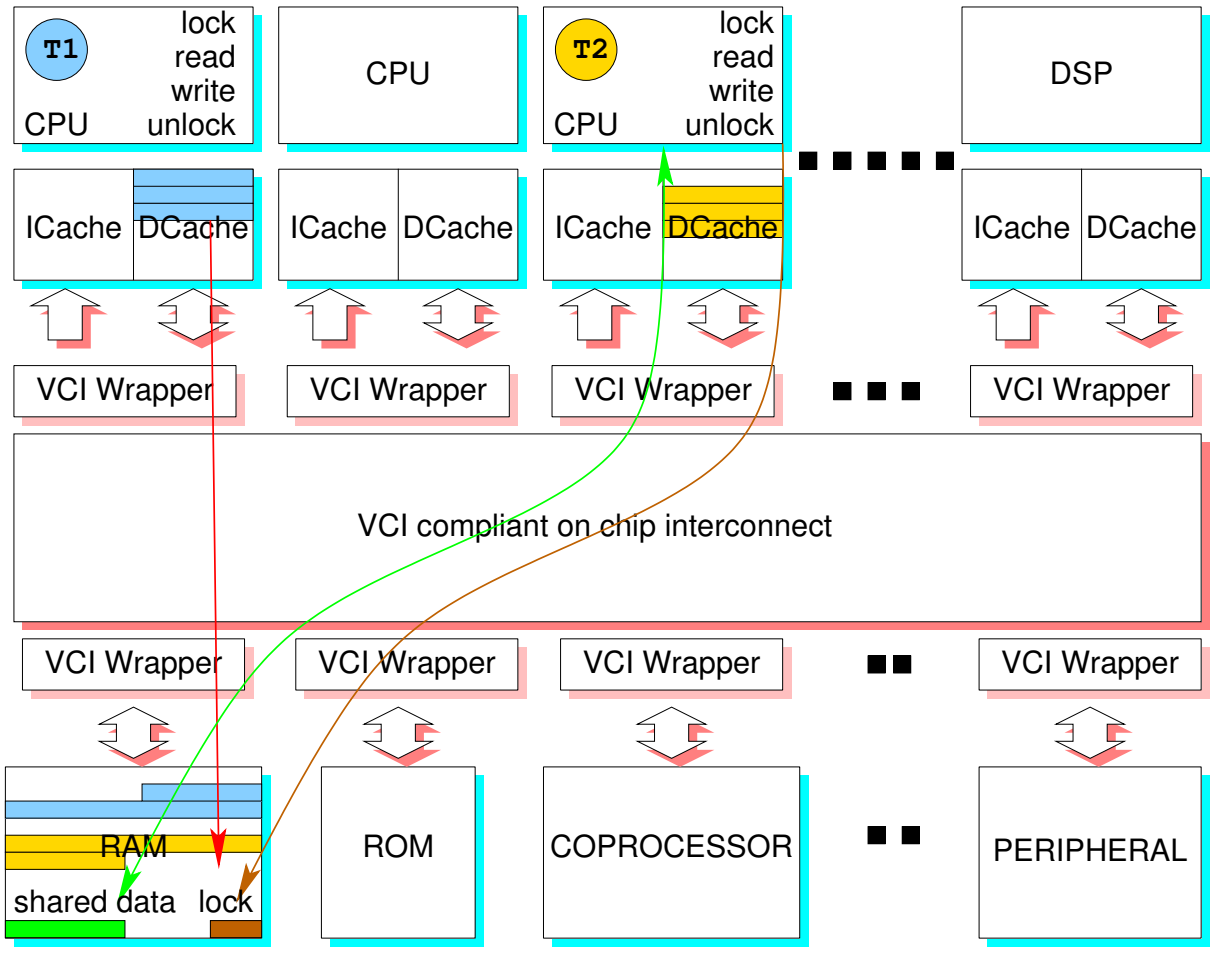
Cohérence matérielle

- $\text{CPI} = 2.88$
- pas de contraintes sur le programmeur
- tâches peuvent migrer sans vidanger les caches

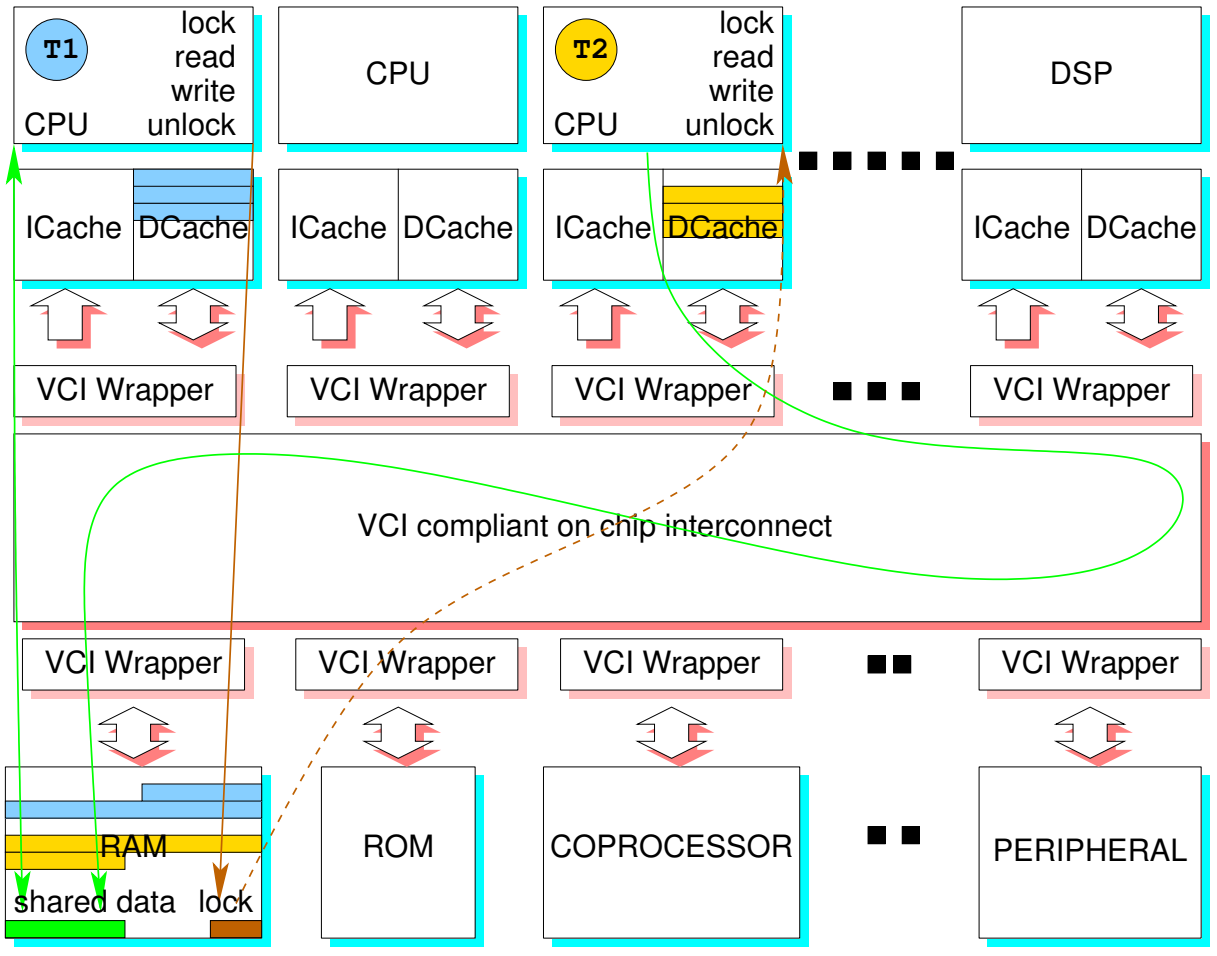
Problème : Consistance mémoire



Problème : Consistance mémoire



Problème : Consistance mémoire



Consistance mémoire

- initiateurs garantissent l'ordre des écritures !
  attente de la réponse avant d'émettre la requête suivante
- *a priori* très coûteux **sauf si** :
  1. les sémaphores sont dans un coprocesseur *ad-hoc ;*
  2. tout initiateur accédant à deux cibles *différentes* doit attendre la réponse de la première cible avant d'émettre la deuxième requête
     conserve l'efficacité des paquets lors des accès à une unique cible.

Plan de la présentation

- Généralités
  - Systèmes sur puce vs Systèmes informatiques
  - Architectures cibles
- Une méthode d'intégration d'applications
  - Principes
  - Problèmes
  - Flot de conception et illustration

Flot : outils

Méthode de conception illustré sur l'exemple d'un décodeur vidéo
Étroitement liés pour aller de la fonctionnalité à l'implantation

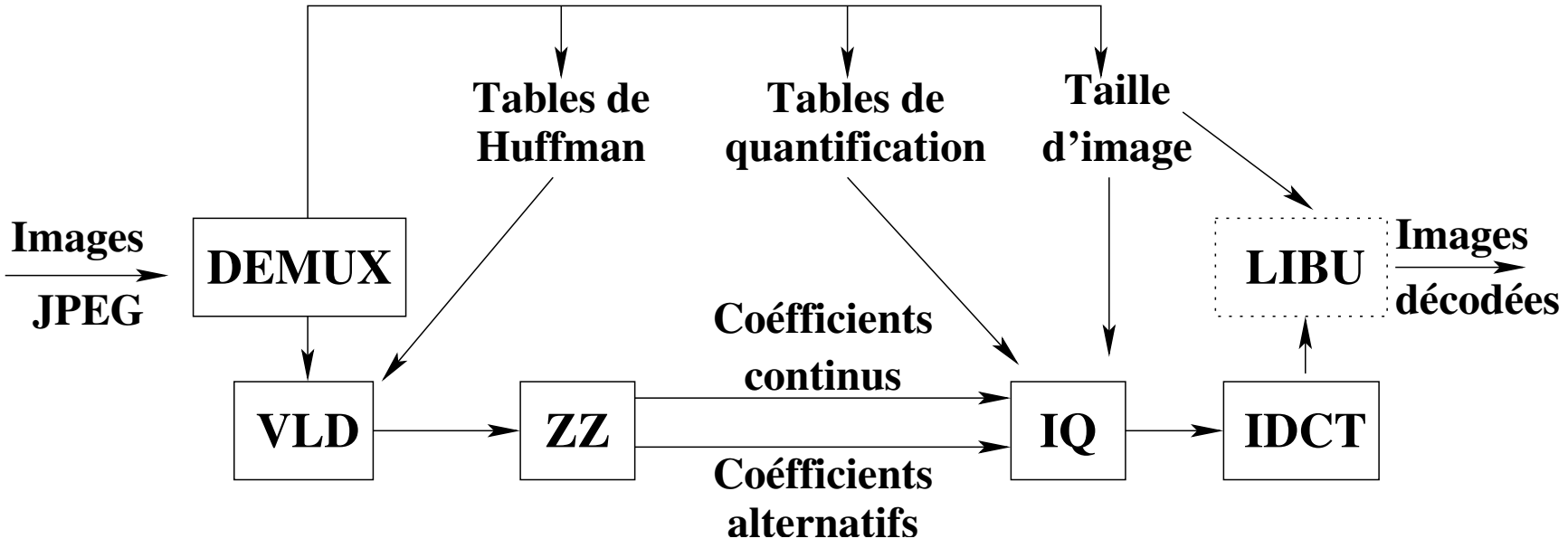
DPN
bibliothèque de fonctions pour les réseaux de Kahn
bâtie au dessus des *pthreads*

CASS
simulation cycle optimisée de modèles C

ASIM0 (PI-Bus)/1 (μ-réseau)
modèles C (simulation) et VHDL (implantation)
Micro-noyau POSIX (Mutek)

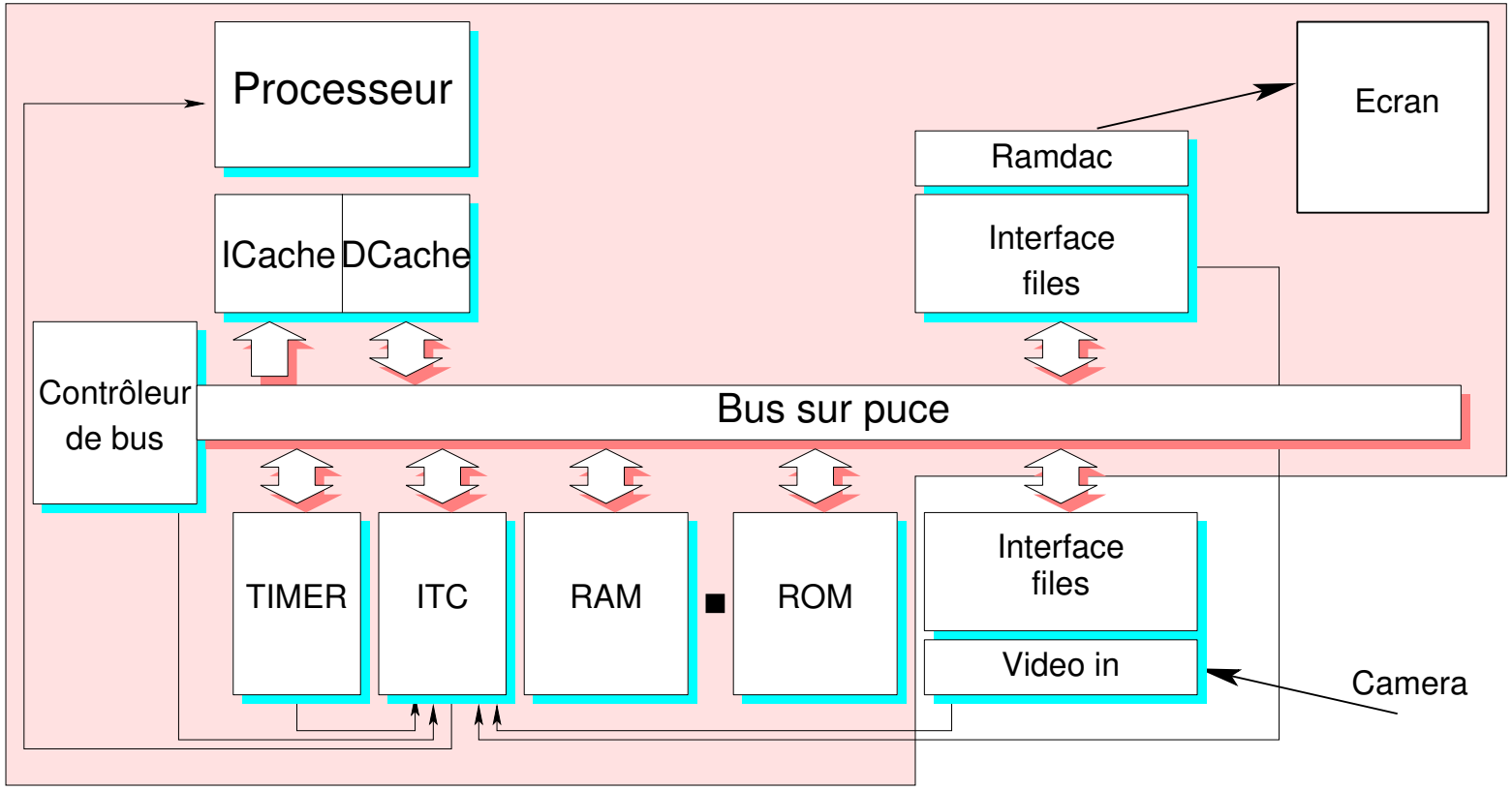
UGH
outil de synthèse d'architecture
produit du C et du VHDL

Flot : décodage *Motion-JPEG*



mjpeg = (**asim0** avec des périphériques d'entrée et de sortie,
décodeur Motion-JPEG,
fréquence de 33 MHz pour 25 images/sec)

Flot : implantation séquentielle



- un seul processeur, compilation séquentielle optimisée
- ajout de périphériques *ad-hoc* pour les E/S
- adaptation des E/S aux périphériques

Flot : implantation séquentielle

Profil séquentiel du Motion-JPEG

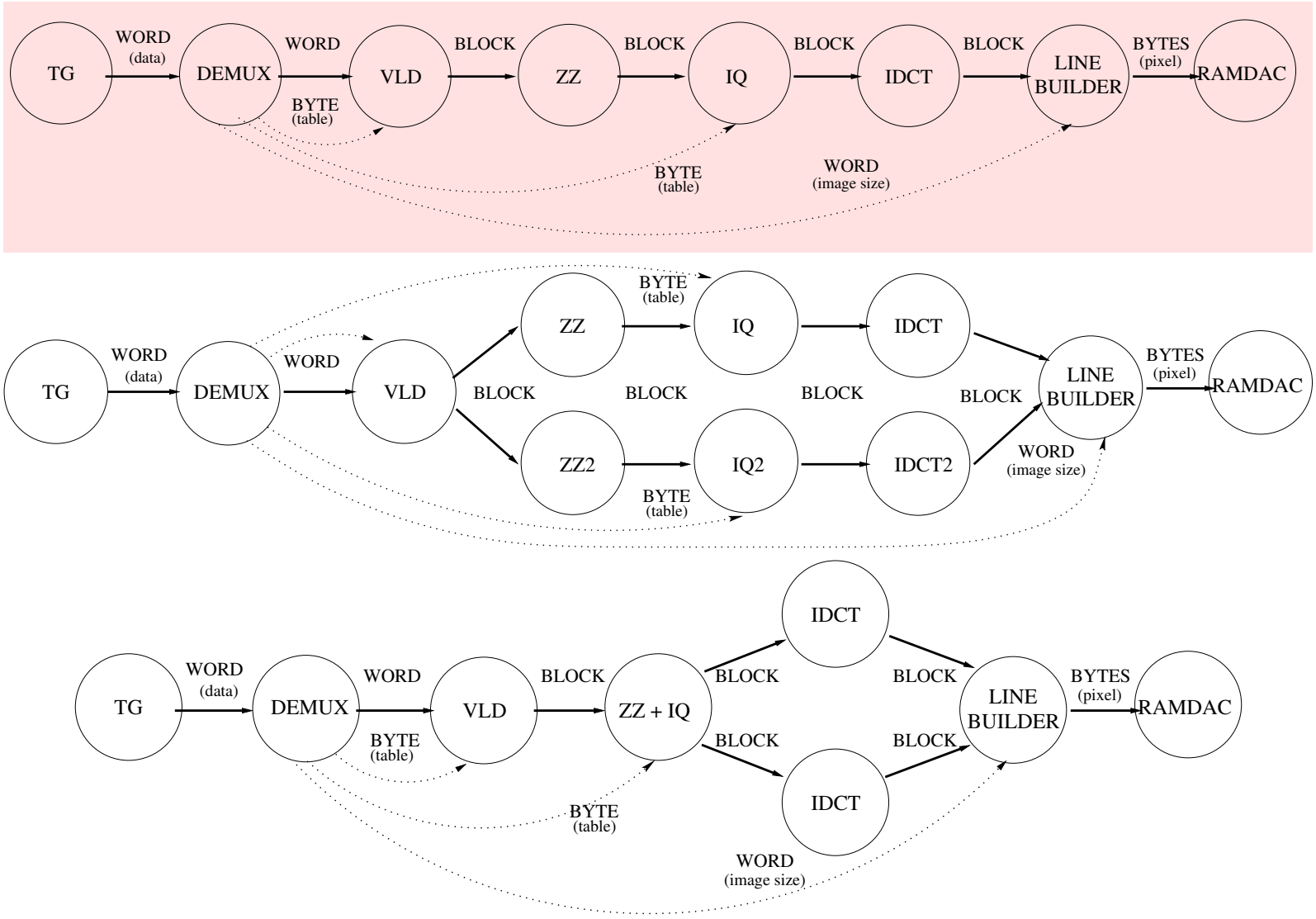
VLD	ZZ + IQ	IDCT	others
23%	22%	46%	9%

Nombre d'octets échangés par image

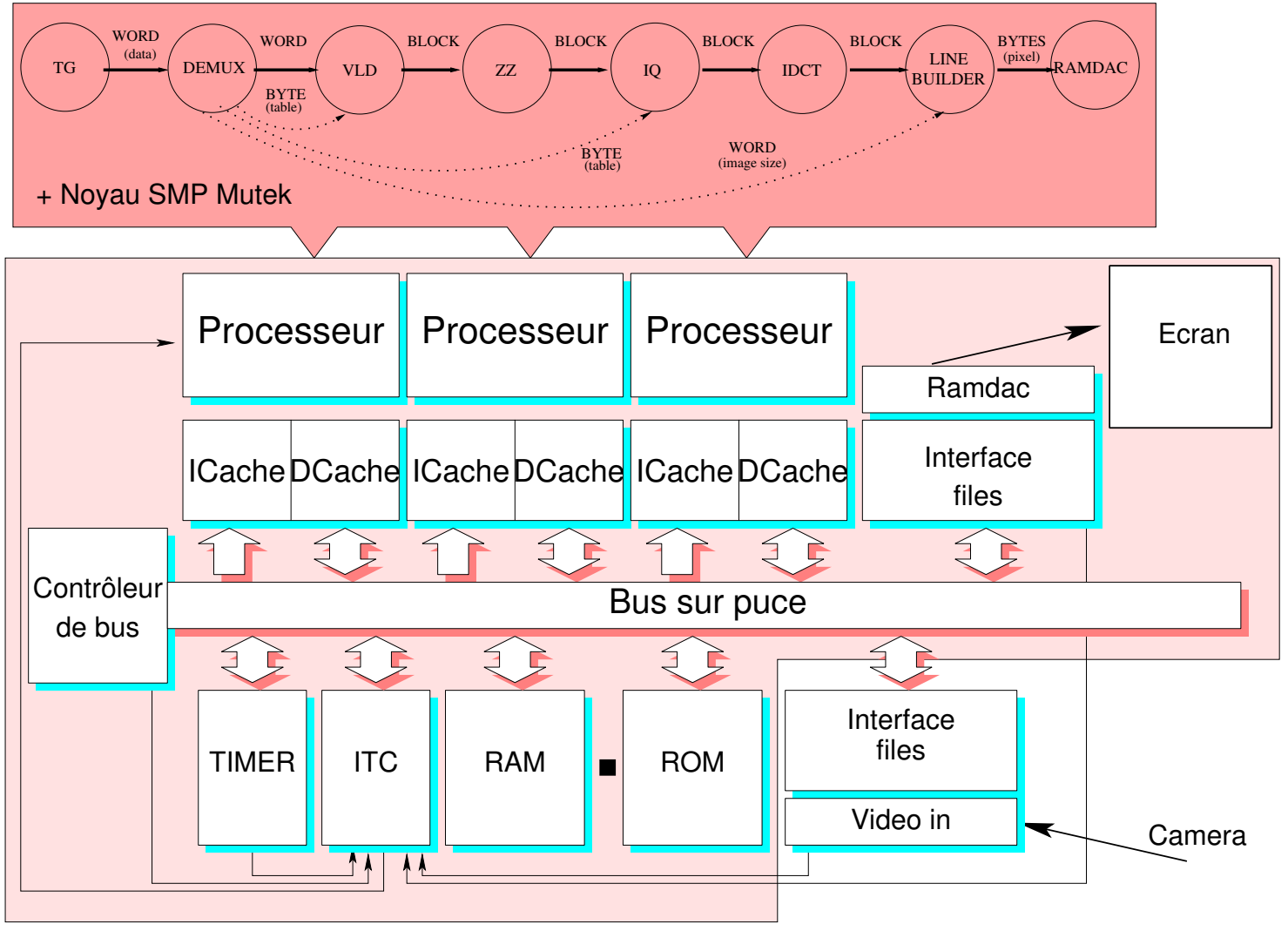
input → DEMUX/ DEMUX → VLD 3024/2691	VLD → ZZ+IQ ZZ+IQ→ IDCT 65536	IDCT → LIBU LIBU → output 16384

Contraintes respectées ⇒ conception achevée
Simulation : 10 images/s ⇒ facteur $2,5$ nécessaire

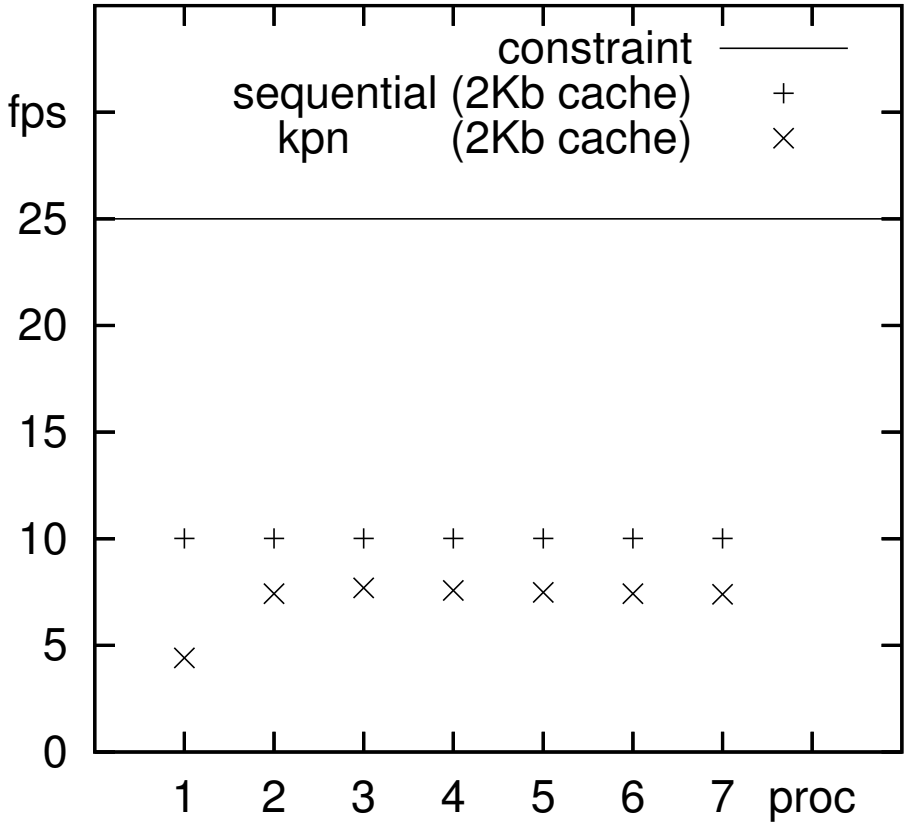
Flot : spécification(s) parallèle(s)



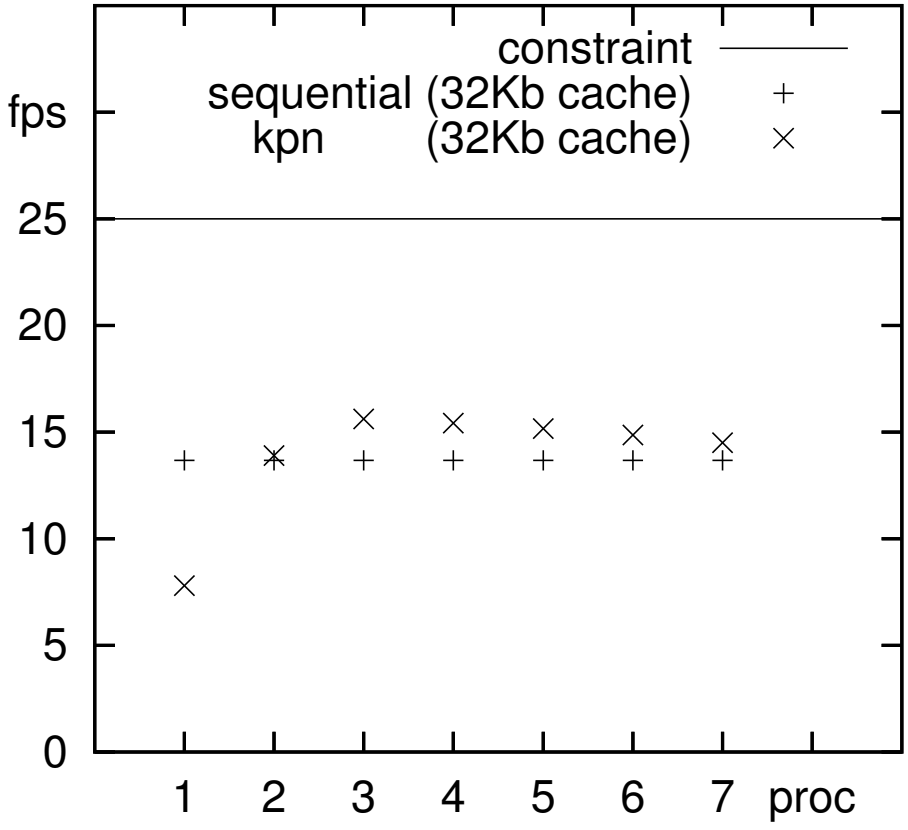
Flot : implantation logicielle parallèle



Flot : implantation logicielle parallèle



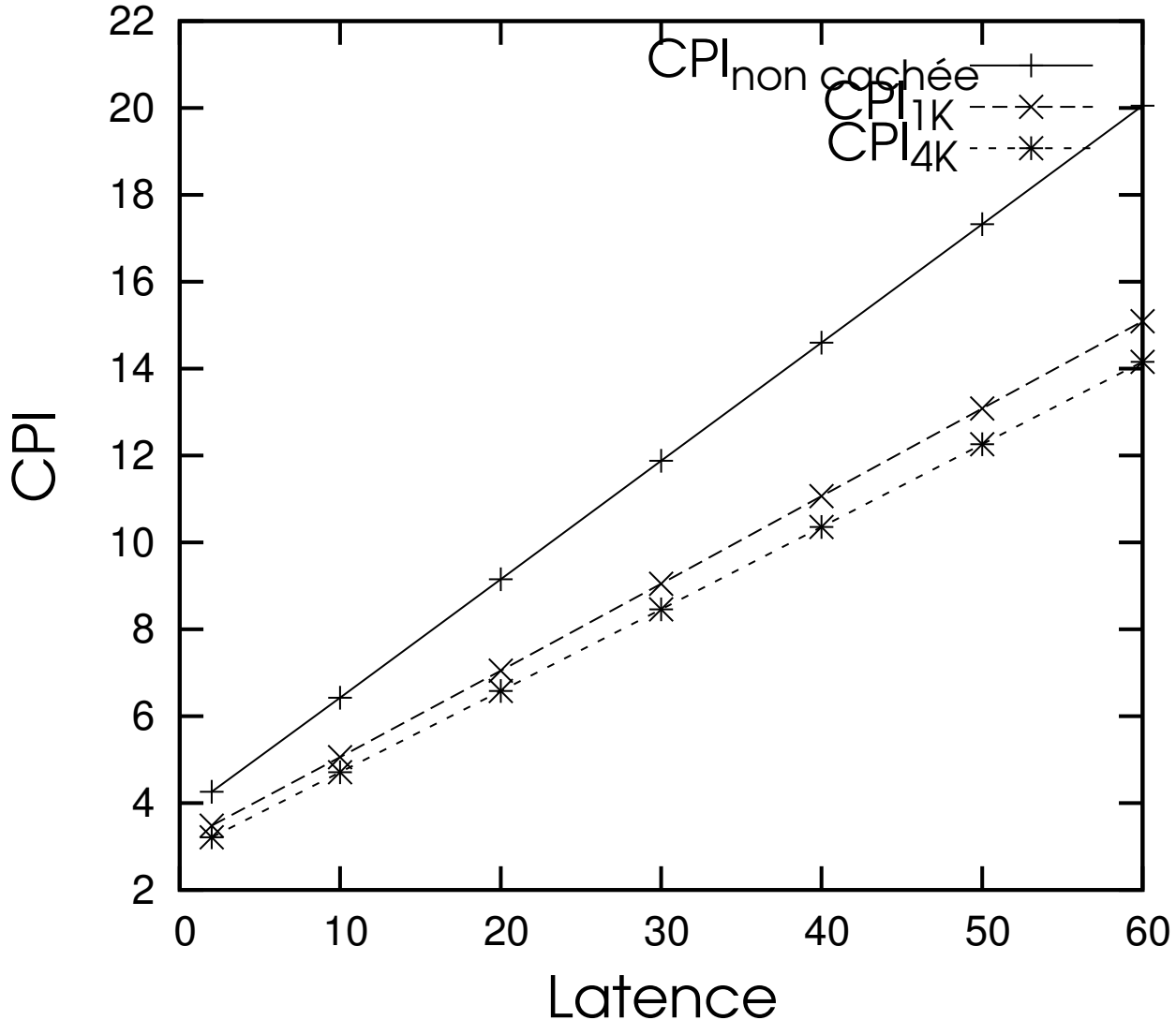
Flot : implantation logicielle parallèle



baisse de performance ($\approx \frac{1}{2}$ sur uniprocessseur)

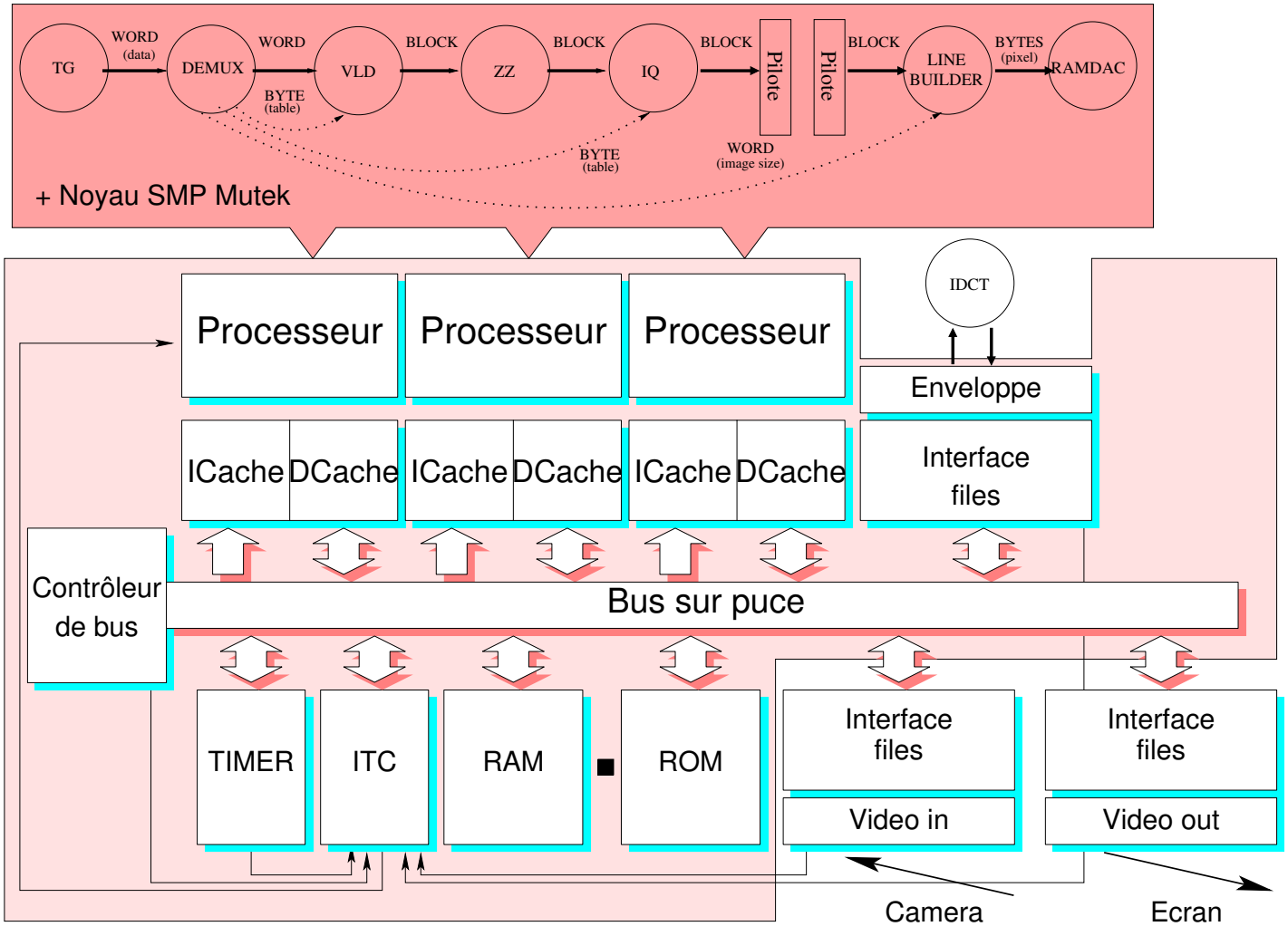
parallélisme à gros grain $\Rightarrow$ tâches matérielles possibles

Flot : Coût de la cohérence



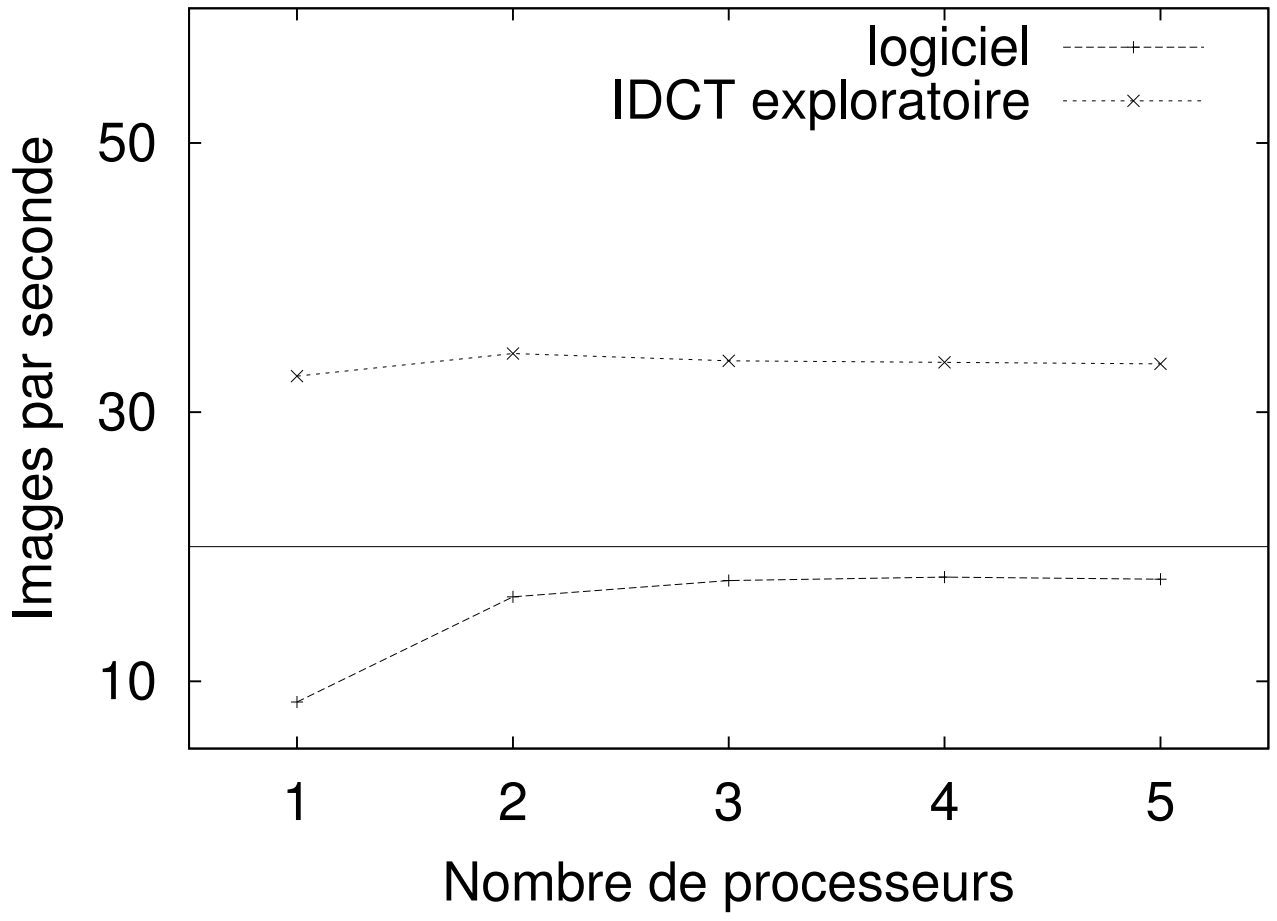
Flot : choix des accélérateurs

Migration exploratoire :



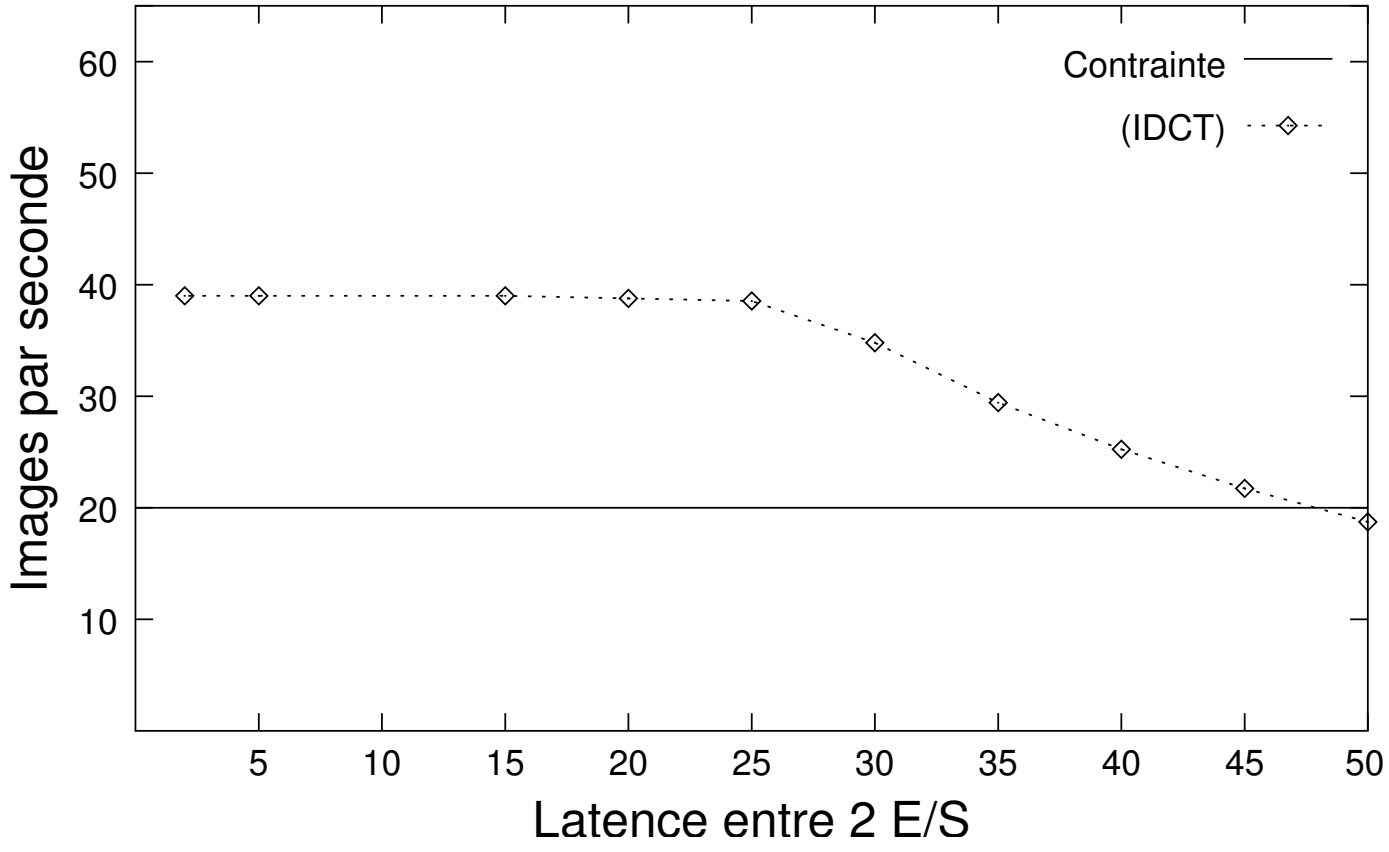
Flot : choix des accélérateurs

Migration exploratoire pour l'IDCT :



Flot : choix des accélérateurs

Migration exploratoire (pour un uniprocesseur) :



Flot : choix des accélérateurs

Contraintes respectées pour :

(IDCT, IQ), 1 processeur
(IDCT, VLD), 1 processeur
(IDCT), 2 processeurs

} si $NW \leq 50$

$\Rightarrow$ Matériel aisément réalisable

Technologie

0.35 μm

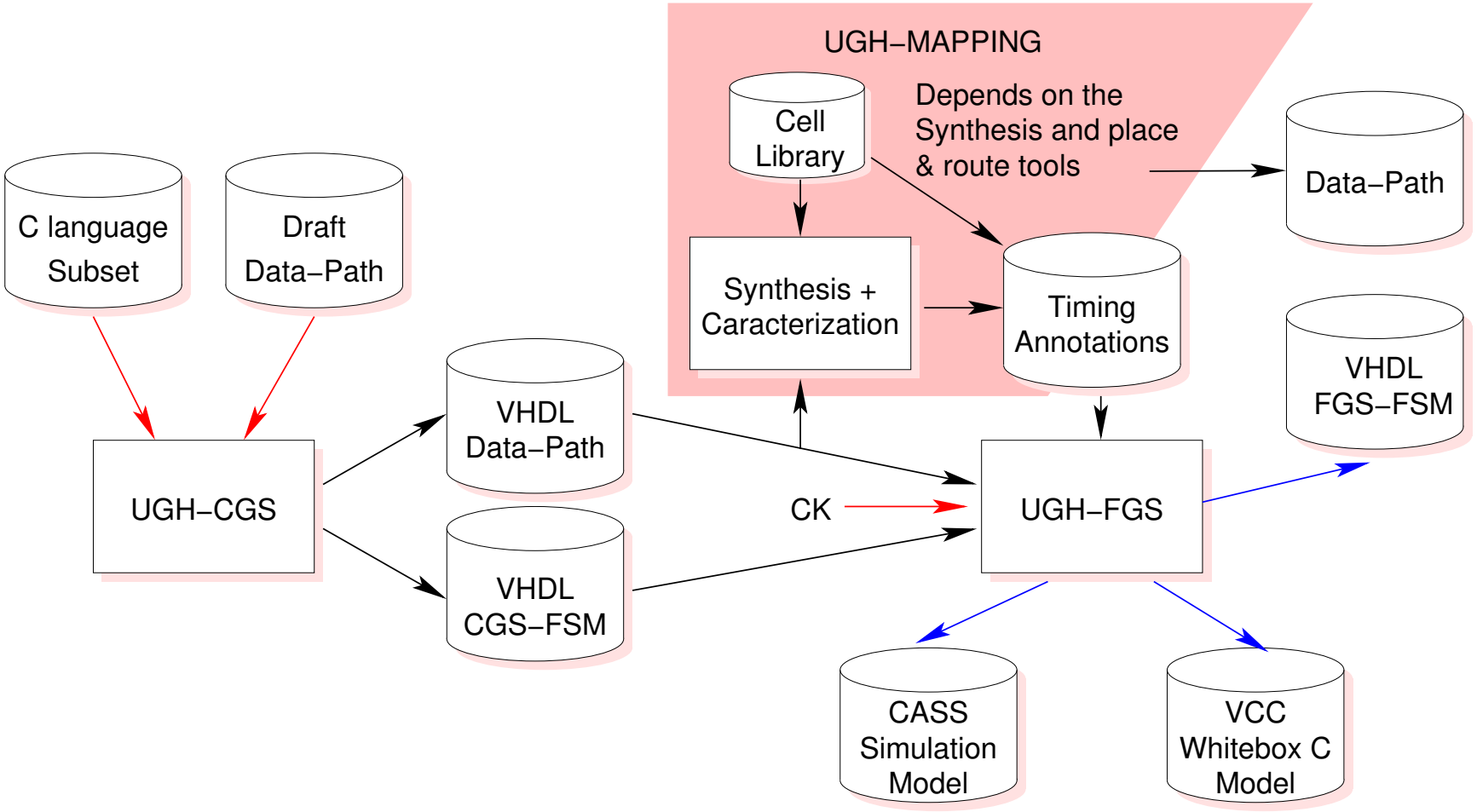
Processeur

Mips R3000 avec 4Ko caches $\Rightarrow \approx 6$ mm^2

Bus

PI-Bus (3-états), fréquence max 133 MHz

Flot : réalisation du matériel



Flot : réalisation du matériel

Particularités :

- guidé par l'utilisateur :
  - opérateurs + registres + interconnexions
  - registres
- fréquence est une entrée :
  - nécessite le chemin de données après synthèse RTL
  - *retiming* d'une machine à états
  - chaînage optimisé d'opérateurs
  - *pipeline*
  - *wave-pipeline*

Flot : réalisation du matériel

Performance de UGH

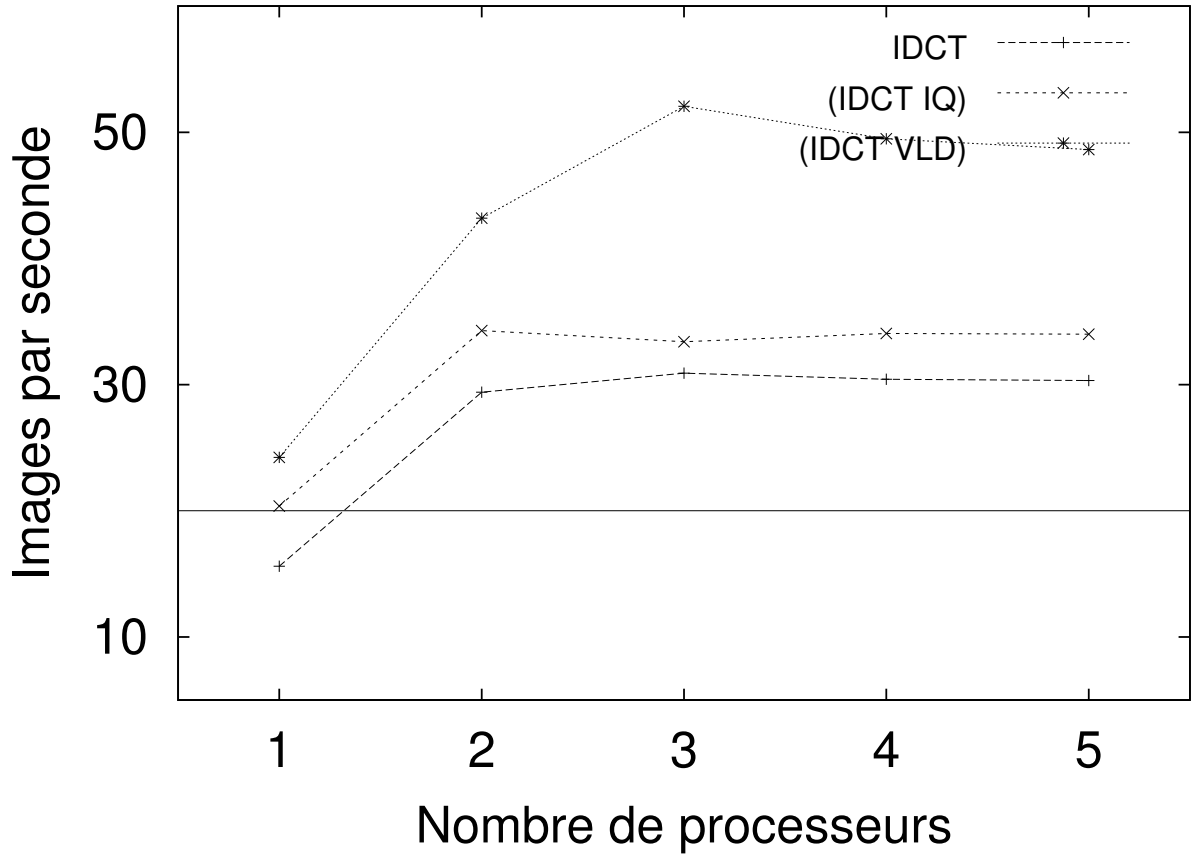
	lignes de C	UGH	calcul des délais somme = *UGH* + *Synopsys*
VLD	200	56.1s	126m41s = 4m40 + 122m01s
IQ	100	0.9s	10m25s = 0m05 + 10m20s
IDCT	600	11.2s	80m50s = 0m54 + 79m56s

Taille des circuits en mm^2

IDCT	VLD	IQ
10.9	**13.2**	**1.2**

Flot : choix des accélérateurs

Migration réelle pour 3 groupes :



Choix : (IDCT, IQ)

Conclusion

Intégration d'applications :

- problème fortement multidisciplinaire
  - architecture, micro-architecture, communication
  - logiciel de bas niveau, noyau, pilotes
  - programmation intrinsèquement parallèle
- problèmes ouverts
  - spécifications, langages, domaines
  - architectures
  - vérification, validation
  - compilation
  - ...