Exploration de l'espace des architectures pour des systèmes de traitement d'image

Analyse faite sur des blocs fondamentaux de la rétine numérique

Rosilde Corvino

14 Octobre 2009

Jeanny HERAULT	Directeur de thèse	GIPSA-lab
Stephane MANCINI	Co-encadrant	GIPSA-lab
Roberto GUIZZETTI	Co-encadrant	STmicroelectronics

Plan

Contexte

Dans le cadre du traitement d'image, nous nous intéressons à:

- la synthèse de haut niveau
- le problème du stockage et du transfert des données

pour des architectures matérielles manipulant un grand nombre de données

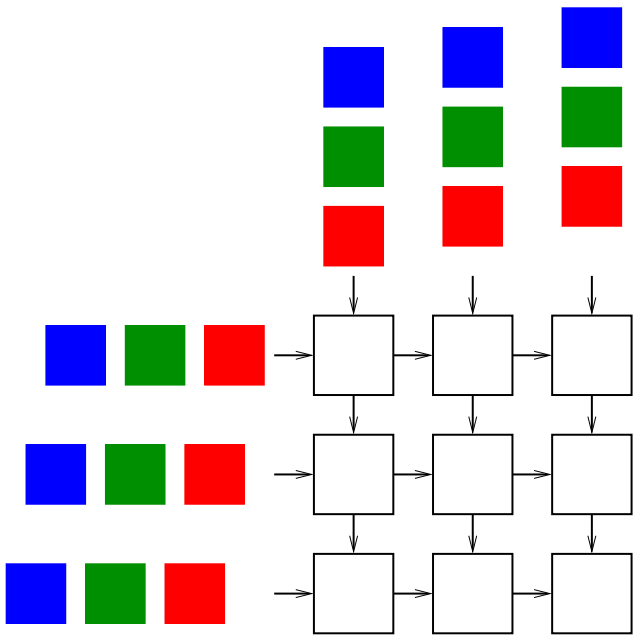
Plan

Systèmes de traitement d'image

Description algorithmique:
ensemble de boucles
(langage type C)

L0: for $p \in \mathtt{D}^n$
S: $I_{t_0}(p) = f_0(W(p))$
end
...
Ln: for $p \in \mathtt{D}^n$
S: $I_{t_n}(p) = f_n(W(p))$
end

Architecture parallèle adaptée:
réseau systolique

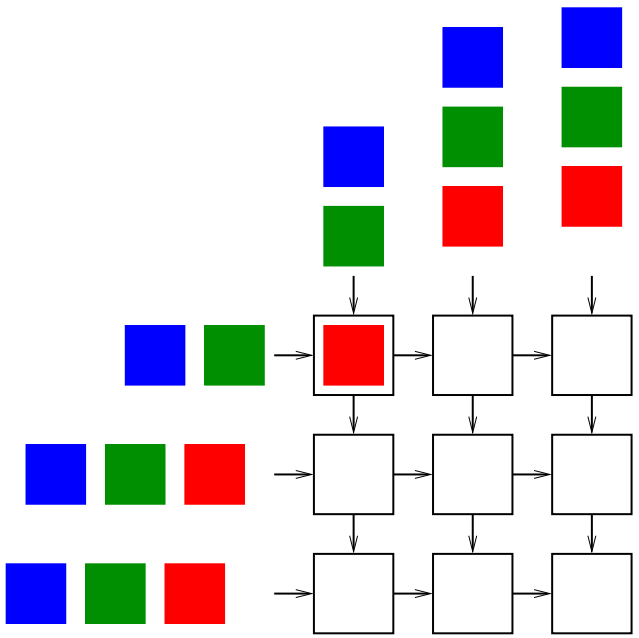


Systèmes de traitement d'image

Description algorithmique:
ensemble de boucles
(langage type C)

L0: for $p \in \mathrm{D}^n$
S: $I_{t_0}(p) = f_0(W(p))$
end
...
Ln: for $p \in \mathrm{D}^n$
S: $I_{t_n}(p) = f_n(W(p))$
end

Architecture parallèle adaptée:
réseau systolique

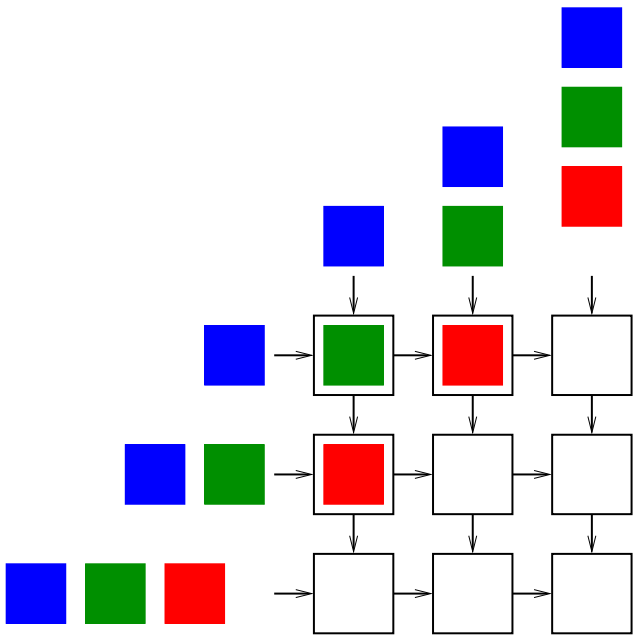


Systèmes de traitement d'image

Description algorithmique:
ensemble de boucles
(langage type C)

L0: **for** $p \in \mathtt{D}^{\mathtt{n}}$
S: $I_{t_0}(p) = f_0(W(p))$
end
...
Ln: **for** $p \in \mathtt{D}^{\mathtt{n}}$
S: $I_{t_n}(p) = f_n(W(p))$
end

Architecture parallèle adaptée:
réseau systolique

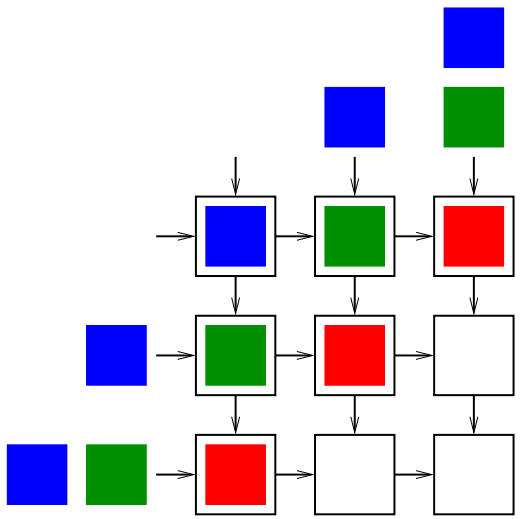


Systèmes de traitement d'image

Description algorithmique:
ensemble de boucles
(langage type C)

L0: for $p \in \mathtt{D}^{\mathtt{n}}$
S: $I_{t_0}(p) = f_0(W(p))$
end
...
Ln: for $p \in \mathtt{D}^{\mathtt{n}}$
S: $I_{t_n}(p) = f_n(W(p))$
end

Architecture parallèle adaptée:
réseau systolique

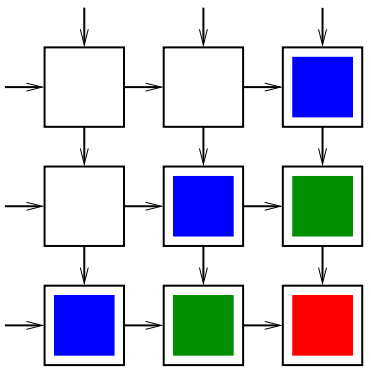


Systèmes de traitement d'image

Description algorithmique:
ensemble de boucles
(langage type C)

L0: for $p \in \mathtt{D}^{\mathtt{n}}$
S: $I_{t_0}(p) = f_0(W(p))$
end
...
Ln: for $p \in \mathtt{D}^{\mathtt{n}}$
S: $I_{t_n}(p) = f_n(W(p))$
end

Architecture parallèle adaptée:
réseau systolique



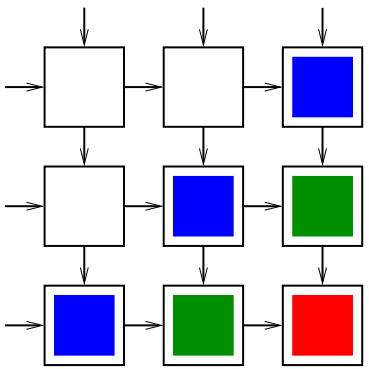
Systèmes de traitement d'image

Description algorithmique:
ensemble de boucles
(langage type C)

L0: **for** $p \in \mathrm{D}^n$
S: $I_{t_0}(p) = f_0(W(p))$
end
...
Ln: **for** $p \in \mathrm{D}^n$
S: $I_{t_n}(p) = f_n(W(p))$
end

Transformation du modèle algorithmique en modèle structural

Architecture parallèle adaptée:
réseau systolique



Synthèse de haut niveau

Definition:

> La synthèse de haut niveau permet de passer d'un modèle algorithmique séquentiel et non-timé vers un modèle structural parallèle et timé

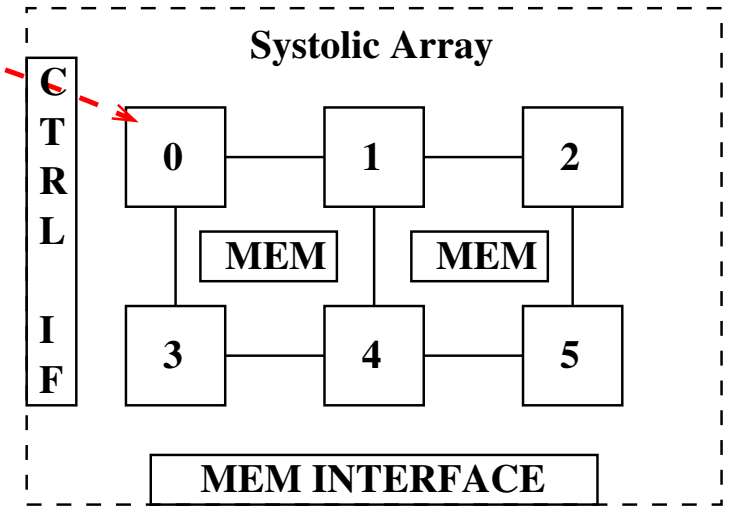
Avantages:

- Augmentation de la **productivité**
- Réduction du **nombre d'erreurs** dans la création du modèle RTL (Register Transfer Level)
- **Temps de simulation** des modèles de haut niveau plus courts
- **Explorer d'avantage de réalisations matérielles différentes**
- ...

Outils de synthèse de haut niveau C-to-RTL

Outil adapté au traitement d'image[1]

```
int SYS(){
  ...
  L0: for(i=1;i<N;i++){
        for(j=1;j<M;j++){
          ...
        }
      }
  ...
  L5: for(i=1;i<N;i++){
        ...
}
```



Étapes de la synthèse de haut niveau

Synthèse Comportementale

Analyse et Opt. du code
Placement des itérations
Gén. du contrôle de boucle

Synthèse Structurale

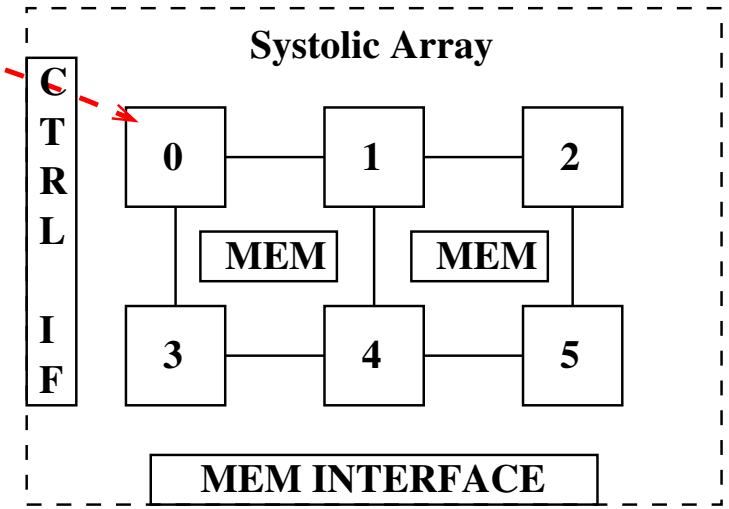
Allocation des opérateurs
Placement des opérations
Gén. du chemin de données

[1] R. Schreiber, The J. of VLSI Signal Processing, 127–142, vol. 31, 2002

Outils de synthèse de haut niveau C-to-RTL

Outil adapté au traitement d'image[2]

```
int SYS(){
  ...
  L0: for(i=1;i<N;i++){
        for(j=1;j<M;j++){
          ...
        }
      }
  ...
  L5: for(i=1;i<N;i++){
        ...
}
```



Étapes de la synthèse de haut niveau

Synthèse Comportementale

Analyse et Opt. du code
Placement des itérations
Gén. du contrôle de boucle

Synthèse Structurale

Allocation des opérateurs
Placement des opérations
Gén. du chemin de données

La méthode de stockage et de transfert de données doit être décidée en amont par l'utilisateur

[2] R. Schreiber, The J. of VLSI Signal Processing, 127–142, vol. 31, 2002

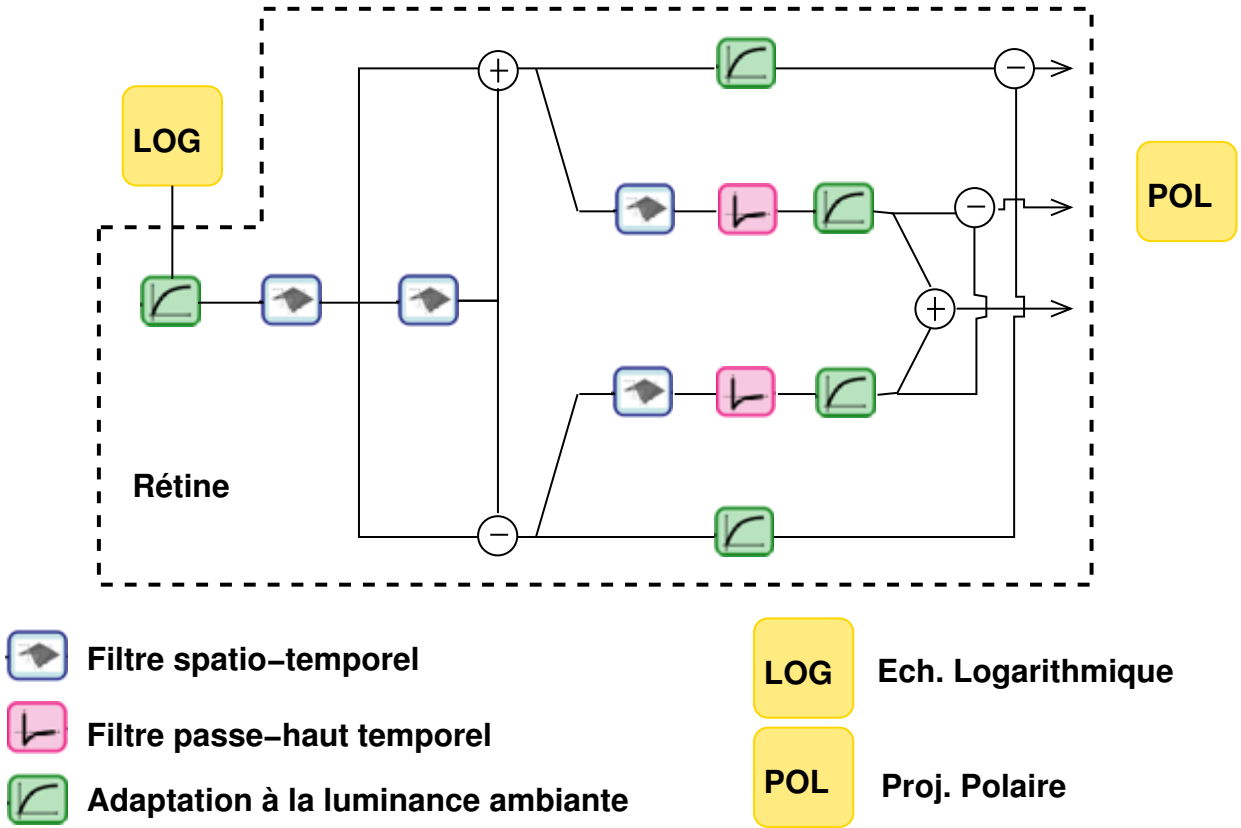
Stockage et transfert de données (1)

Problème concernant les algorithmes gourmands en calculs et traitant un grand nombre de données

1. **Stockage des données en mémoire interne** $\Rightarrow$ Augmentation de la surface occupée par le circuit
2. **Transfert des données de la mémoire externe (DRAM)** $\Rightarrow$ Dégradation des performances temporelles du système à cause de la latence mémoire

Stockage et transfert de données (2)

Cas de la rétine numérique [3]



17 images intermédiaires	#300 op. par pixel	15 images par sec.

[3] J. Hérault, IWANN, 662-675, 2007

Stockage et transfert de données (3)

Cas de la rétine numérique [4]

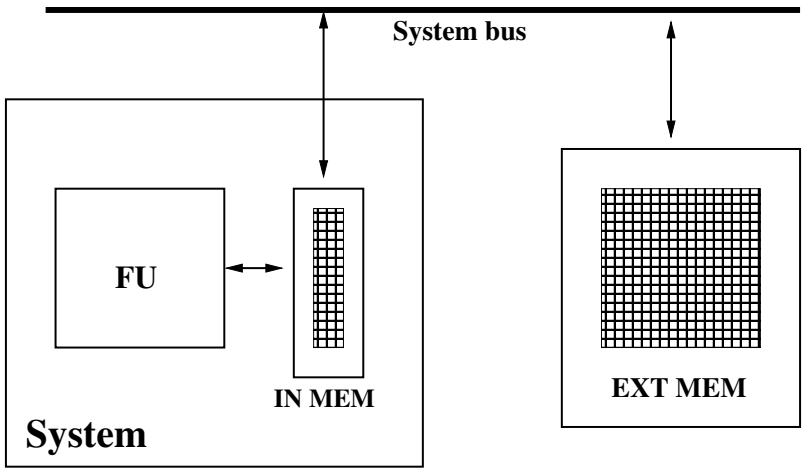
Cas hypothétique: toutes les images intermédiaires en interne			
	Image d'entrée	**Est. mémoire interne** (SRAM - mm^2)	**Complexité** (MOPS)
SQCIF	128×96	1,6	55,5
VGA	640×480	40	1383
1080i	1920×1080	276	9331,2
Application industrielle réelle (tout le système) $\Rightarrow$ 20 mm^2			

Compromis entre le stockage des données en interne et le transfert depuis la mémoire externe:

Transformations de l'**architecture matérielle** et de l'**algorithme**

[4] J. Hérault, IWANN, 662-675, 2007

Solution architecturale: créer une hiérarchie mémoire



Étapes d'une **exploration** des hiérarchies mémoire possibles[5]:

- **Estimation et allocation** de la quantité de mémoire nécessaire
- **Assignation** des tableaux de données aux différentes mémoires internes

[5] Panda, TODAES, 149–206, 2001

Solution algorithmique: partitionnement

Solutions dans la littérature: partitionner les opérations effectuées par l'algorithme et trouver le *footprint* (trace des données accédées) correspondant.

Calcul du *footprint* par:

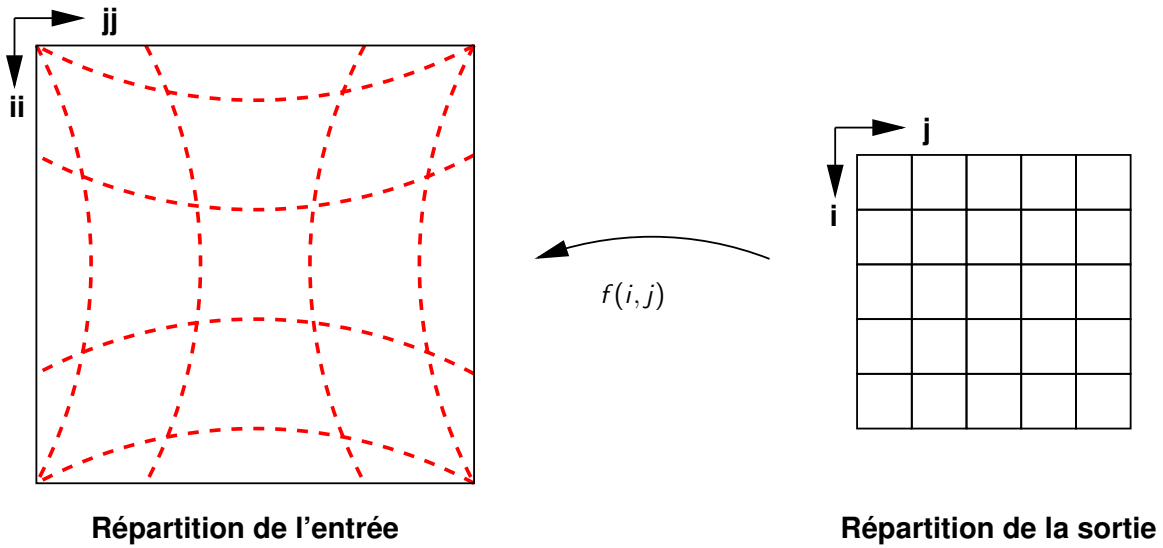
1. **Analyse dynamique**. Construction d'une trace des accès aux données par simulation. **Pas de condition sur les lois d'accès aux données**. Les *Footprints* ont une taille variable.
   - N. Mitchell, Proceedings of PACT, 1999
   - C. Huang, Transaction on VLSI Systems, November 2007

2. **Analyse statique** des dépendances par méthodes de programmation linéaire. **Les lois d'accès aux données doivent être affines**. Les *Footprints* ont une taille constante.
   - Boulet, VLSI Journal, 33-52, 1993
   - Rastello, Transacton on PDS, May 2002

Le partitionnement dans le cas des lois d'accès non-affines

Ex: l'échantillonnage logarithmique (prétraitement de la rétine)

$$(ii, jj) = (\rho i, \rho j) \qquad \rho = \frac{\rho_0}{\rho_{lim} - \sqrt{i^2 + j^2}}$$



Répartition régulière des opérations pour le calcul de la sortie ⇒
Répartition irrégulière des données d'entrée

Méthode non intégrée dans les outils de HLS

Contrôle pour le transfert de quantités de données variables

Le partitionnement dans le cas des lois d'accès affines

Modèle polyèdrique: utile aux transformations de boucles imbriquées[6] et intégré dans les outils de HLS

```
for i = 0 : N
    for j = 0 : M
        Im( i , j )= Im( i-1 , j-1 )
                   +Im( i , j+1 )
    end
end
```

- modèle matriciel, ex.

$$p = \begin{pmatrix} i \\ j \end{pmatrix} \text{ et } D = \begin{pmatrix} -1 & 0 \\ -1 & 1 \end{pmatrix}$$

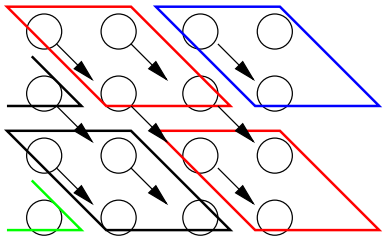
- une transformation de boucle **T**

$$D' = \mathbf{T}D \text{ et } p' = \mathbf{T}p$$

Loop Tiling

```
for Tp ∈ Dⁿ
    S: It(Tp) = f(W(Tp))
end
```

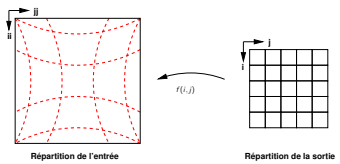
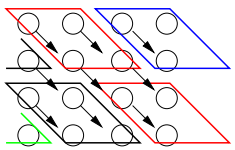
Partitionnement des données



Limitation: Les lois d'indexation des tableaux de données doivent être affines !

[6] Bastoul, Lectures Notes In Computer Science, 209-225, 2004

Conclusion sur les méthodes de partitionnement

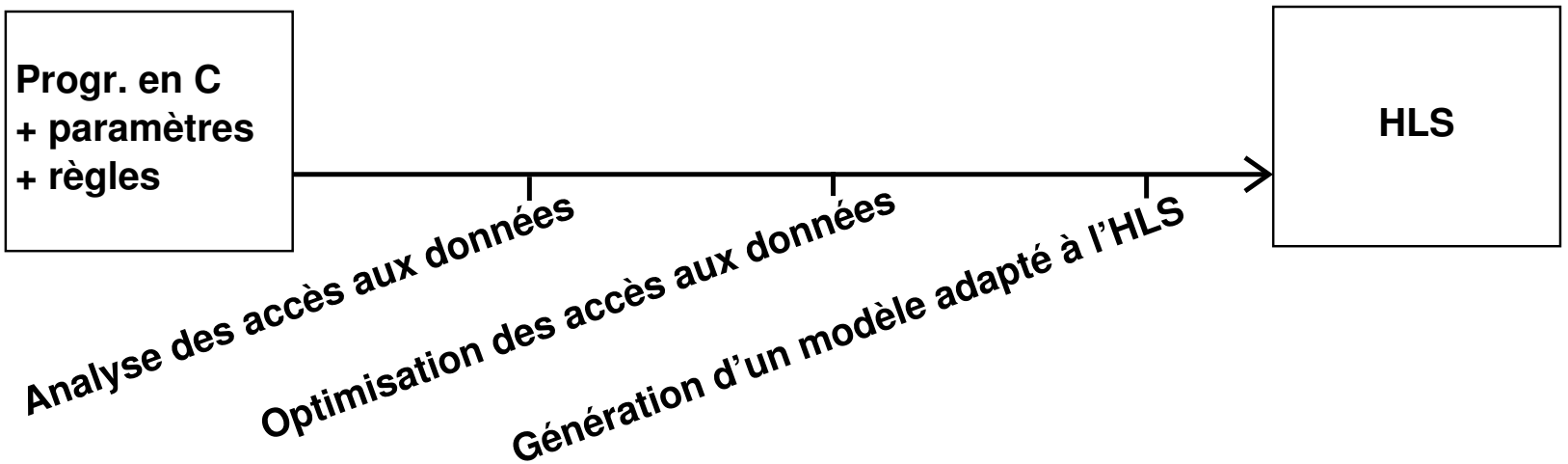
Méthode	Avantages	Inconvénients
1 	* les lois d'accès non-affines	* contrôle "complexe" sur le transfert des données * non intégrée dans les outils de HLS
2 	* contrôle simple sur le transfert des données * intégrée dans les outils de HLS	* les lois d'accès affines

Objectif: trouver une méthode de partitionnement permettant de gérer des accès non-affines (type 1) et minimisant le contrôle sur le transfert des données (type 2)
La méthode doit être adaptée aux outils de HLS.

Objectif du Doctorat

Développer un Outil d'exploration qui

- Optimise les accès mémoire de systèmes parallèles pour le traitement d'image
- Gère les accès non-affines
- Génère un code C synthétisable avec un outil de synthèse de haut niveau

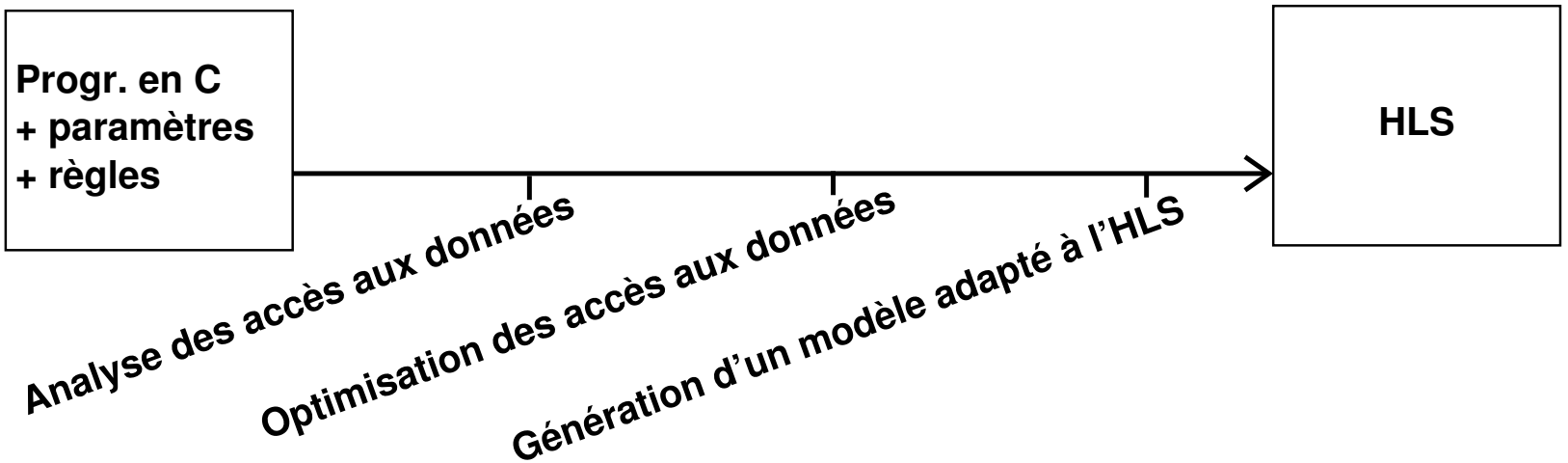


Objectif du Doctorat

Développer un Outil d'exploration qui

- Optimise les accès mémoire de systèmes parallèles pour le traitement d'image
- Gère les accès non-affines
- Génère un code C synthétisable avec un outil de synthèse de haut niveau

MEXP= Memory EXPloration



Plan

Nomenclature

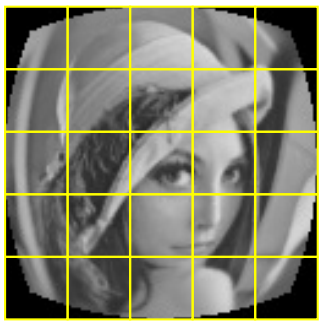
Dans notre méthode nous utilisons la nomenclature suivante:

- **Tiling :** partitionnement
- **Tuile :** bloc de données ou d'itérations
- **Tiling d'entrée :** partitionnement des données d'entrée
- **Tiling de sortie :** partitionnement des itérations nécessaires au calcul de la sortie

Tilings indépendants de l'entrée et de la sortie

Proposer une solution au problème du **partitionnement** pour des algorithmes avec lois d'indexation **non affines**

Entrée

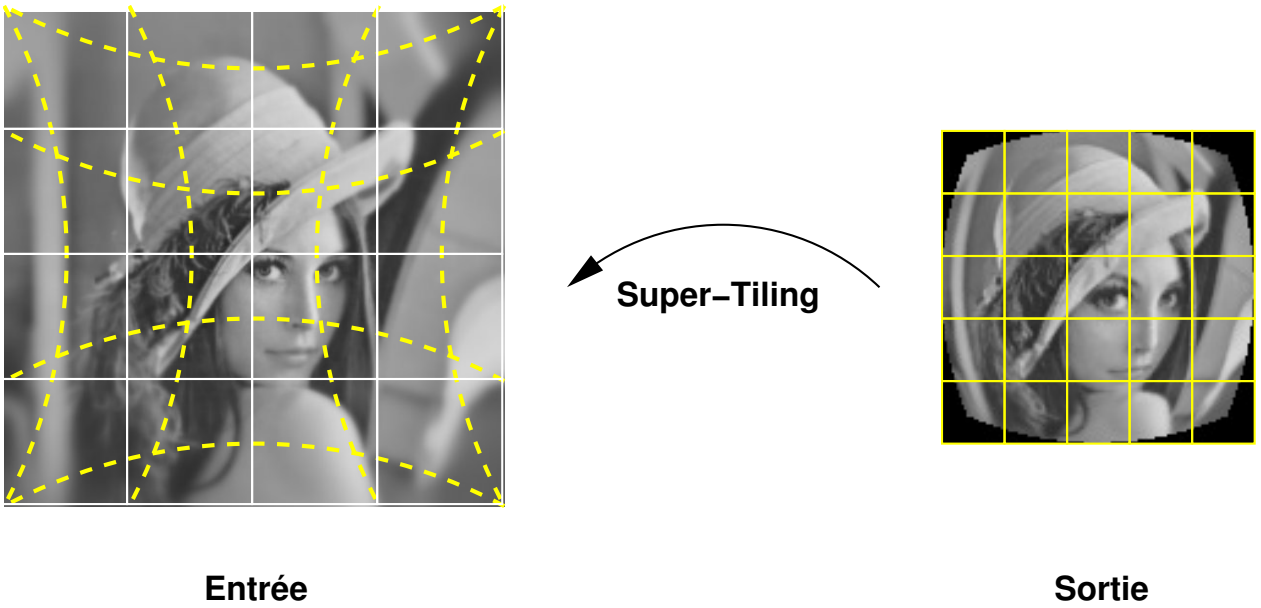


Sortie

Condition pour l'application de la méthode: les algorithmes cibles ont des dépendances statiques

Projeter le tiling de sortie sur le tiling d'entrée

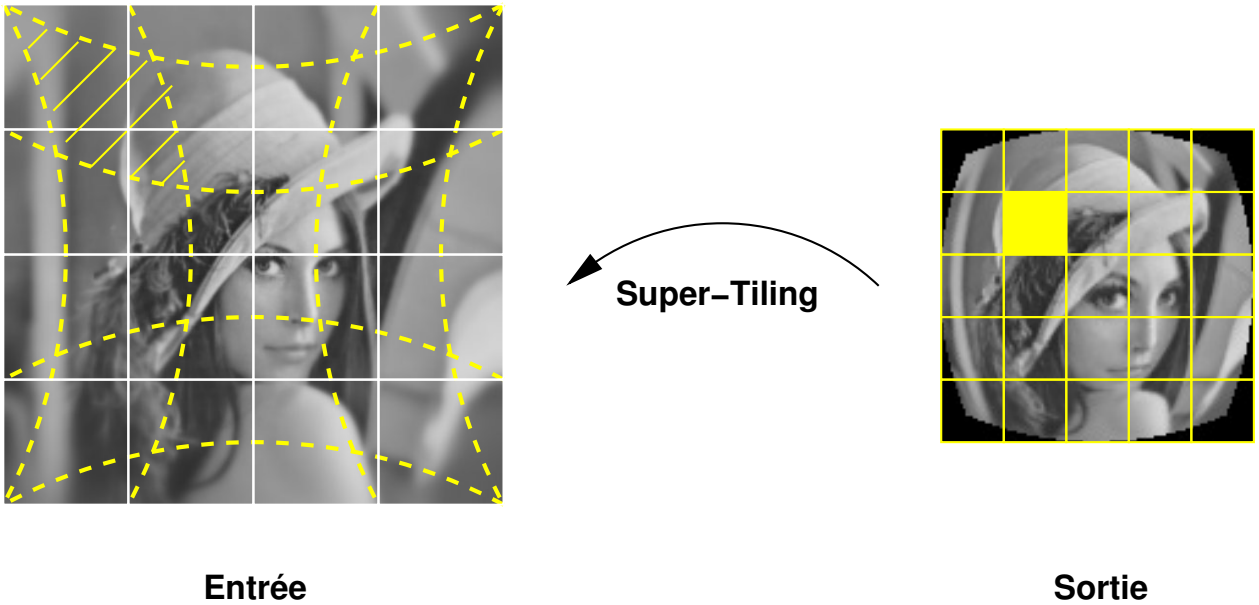
La projection s'effectue selon la loi d'indexation du tableau de données d'entrée



Condition pour l'application de la méthode: les algorithmes cibles ont des dépendances statiques

Selection des tuiles d'entrée utiles

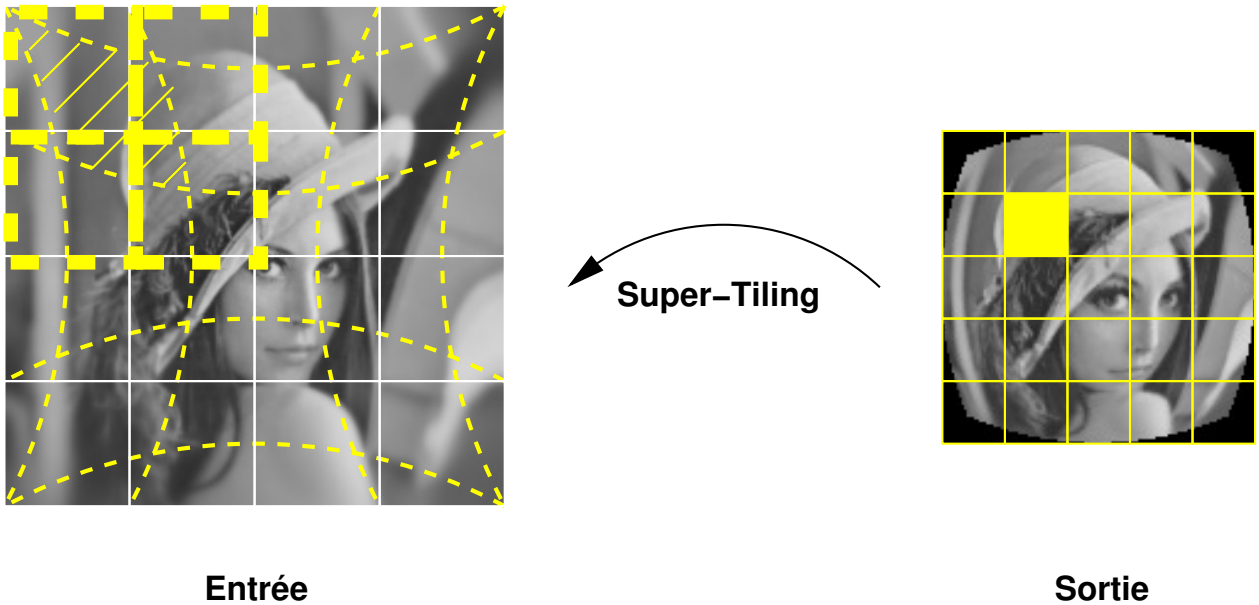
Trouver les tuiles d'entrée qui contiennent la projection des tuiles de sortie dans l'espace d'entrée



Adaptation du tiling à l'algorithme cible, **en explorant** différentes tailles de tuiles d'entrée et de sortie

Selection des tuiles d'entrée utiles

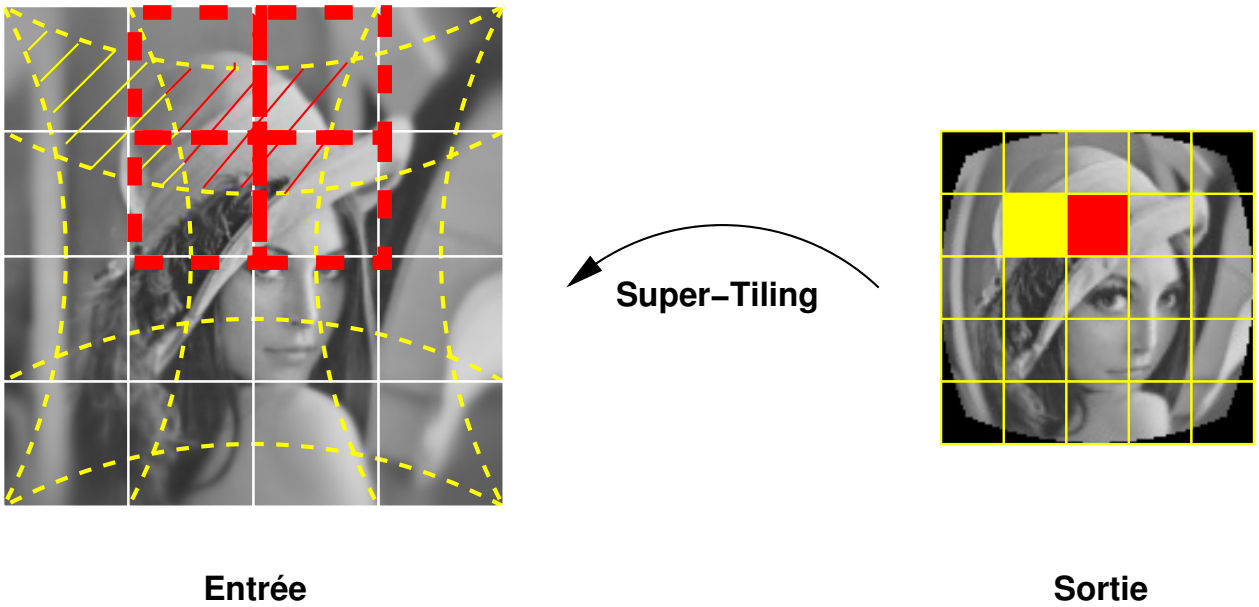
Trouver les tuiles d'entrée qui contiennent la projection des tuiles de sortie dans l'espace d'entrée



Adaptation du tiling à l'algorithme cible, **en explorant** différentes tailles de tuiles d'entrée et de sortie

Ordonnancer les calculs des tuiles de sorties

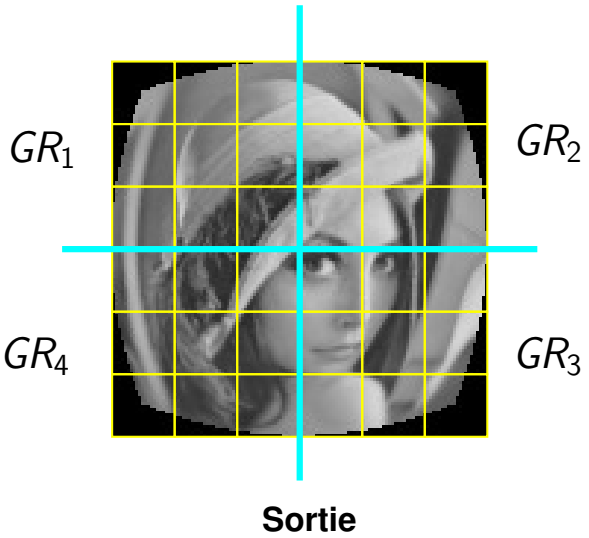
Conditions sur l' algorithme cible: parallèle et avec dépendances statiques



But: optimiser une caractéristique du matériel cible, quantité de mémoire interne utilisée ou performances temporelles

Parallélisme inter-tuiles

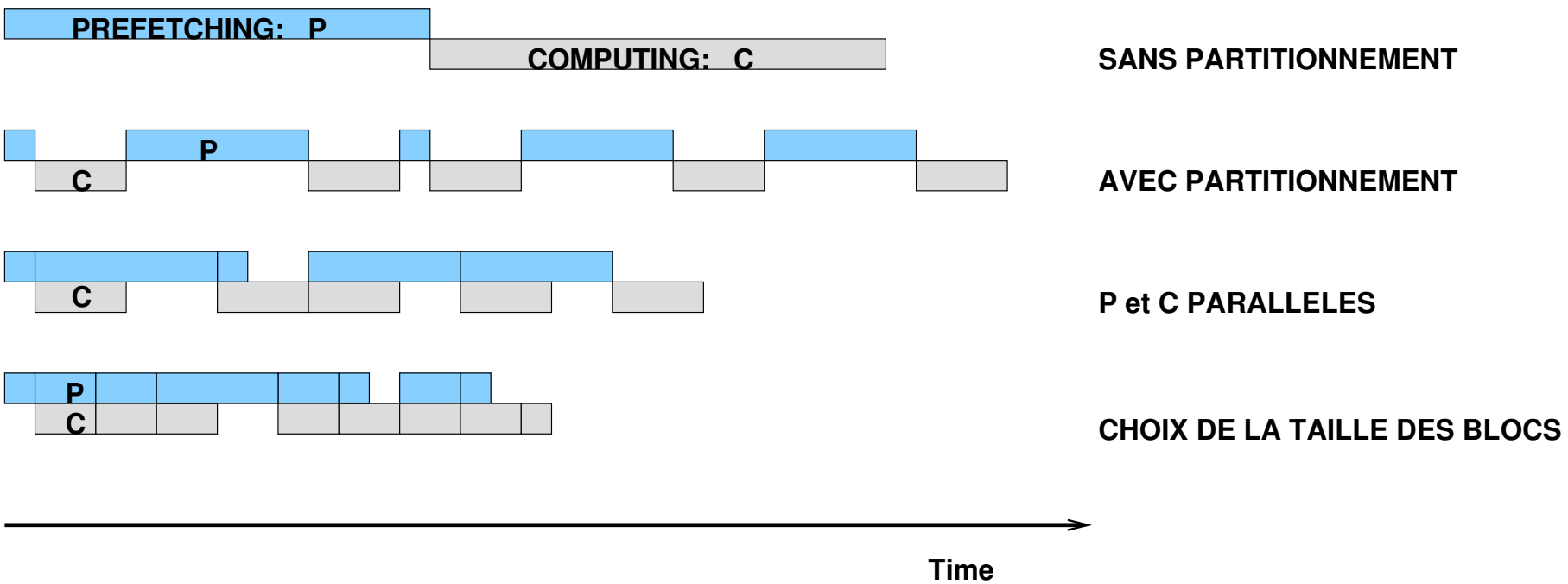
Diviser les tuiles de sortie en groupes indépendants



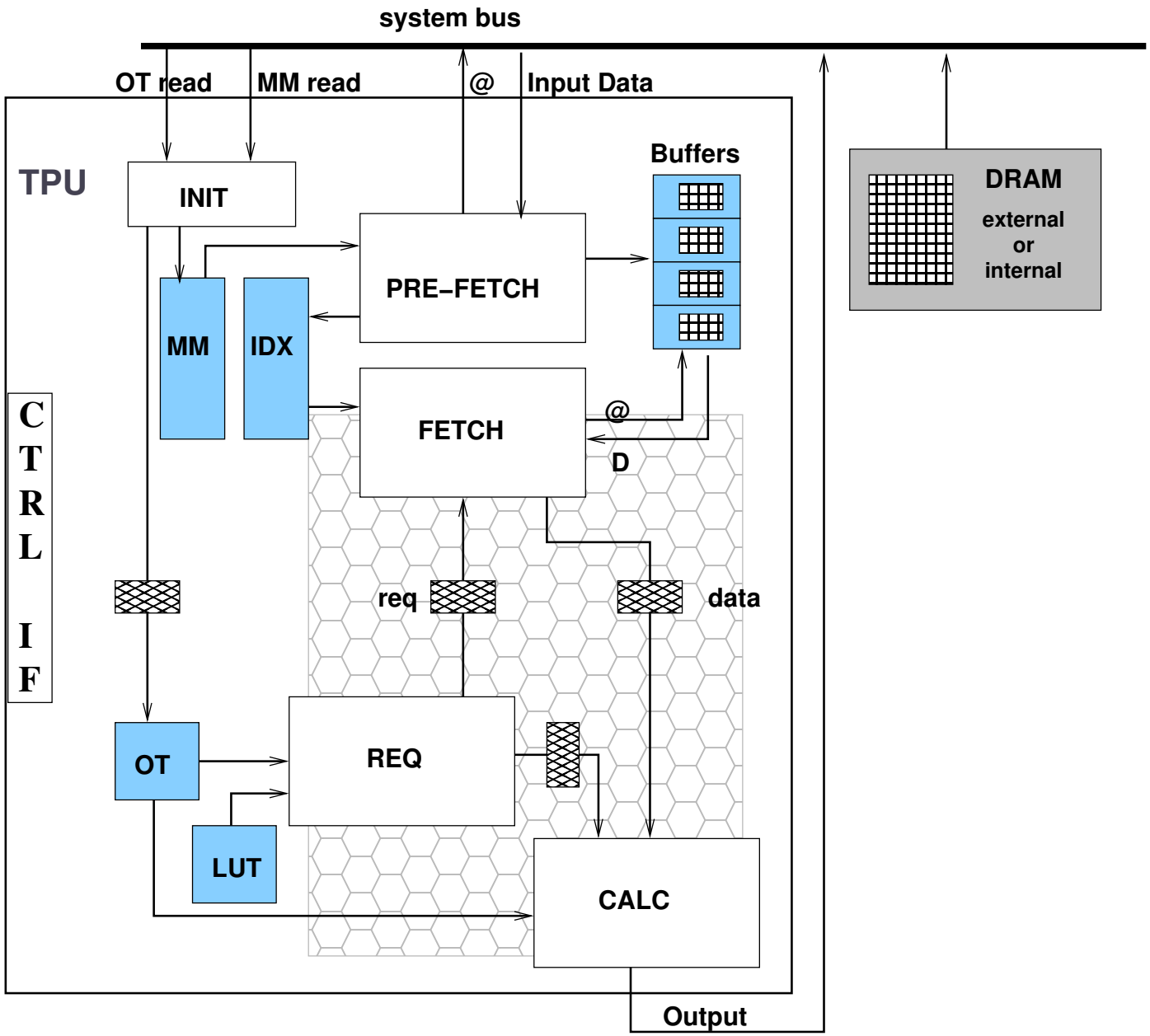
Les tuiles d'un même groupe seront calculées séquentiellement et les différents groupes seront calculés en parallèle

Avantages de la méthode

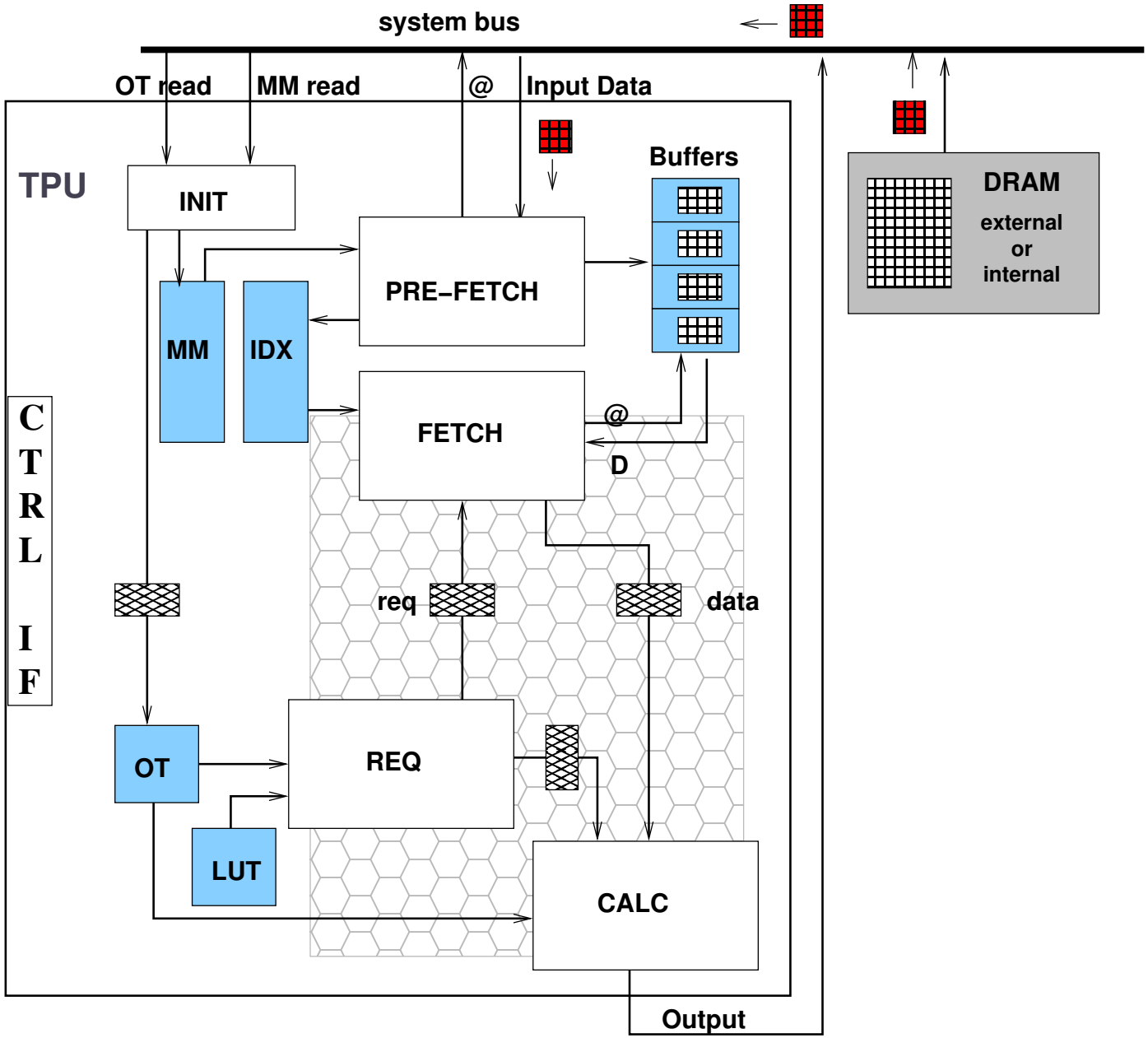
1. Partitionnement: localité des données et réutilisation des ressources mémoire
2. Transfert de quantités variables de données par tuile de taille constante
3. Explorer différentes tailles de tuiles d'entrée et de sortie pour:
   - réduire la mémoire interne
   - masquer le temps de pre-fetch avec le temps de calcul



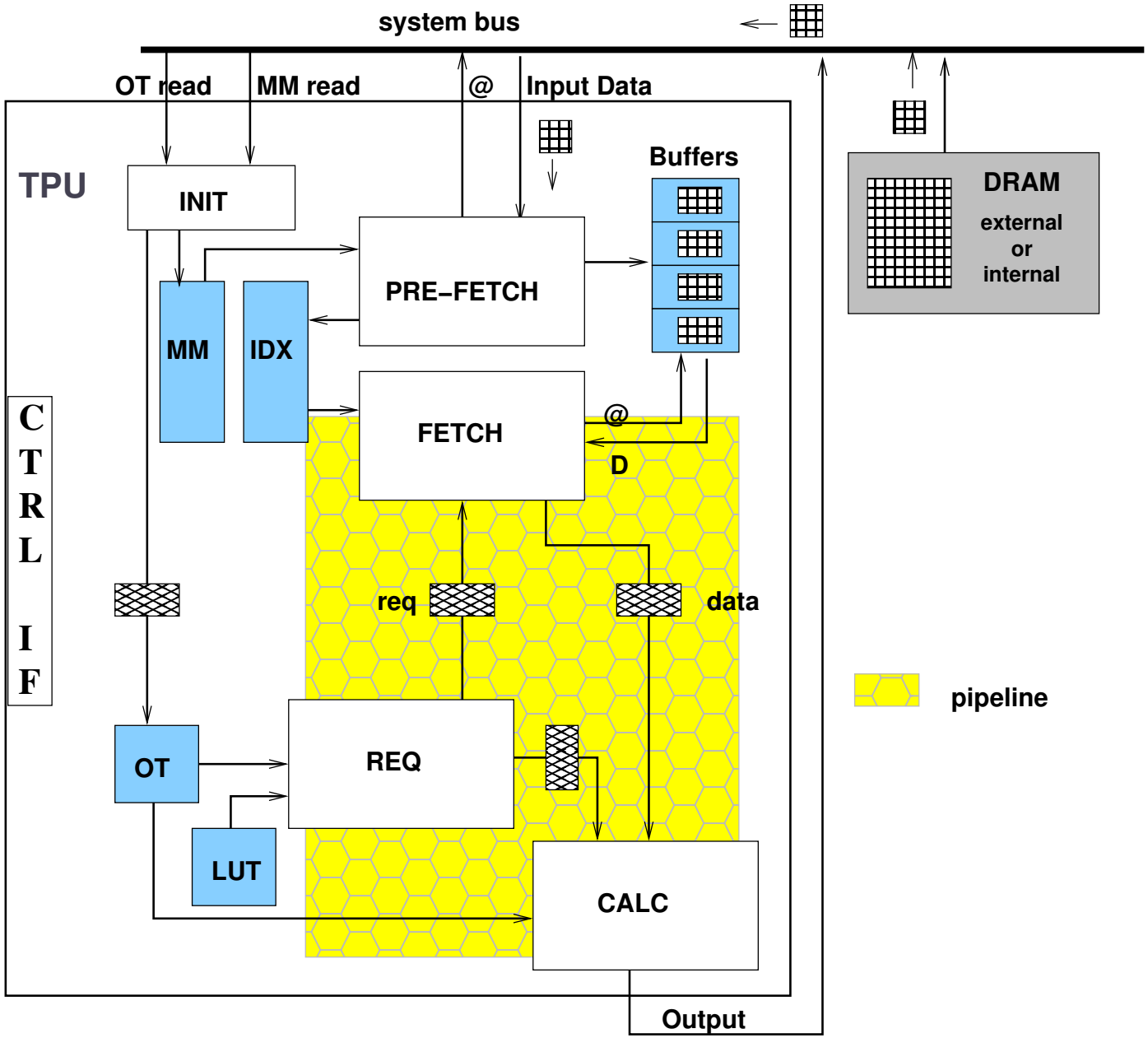
L'architecture cible: le TPU (Tile Processing Unit)



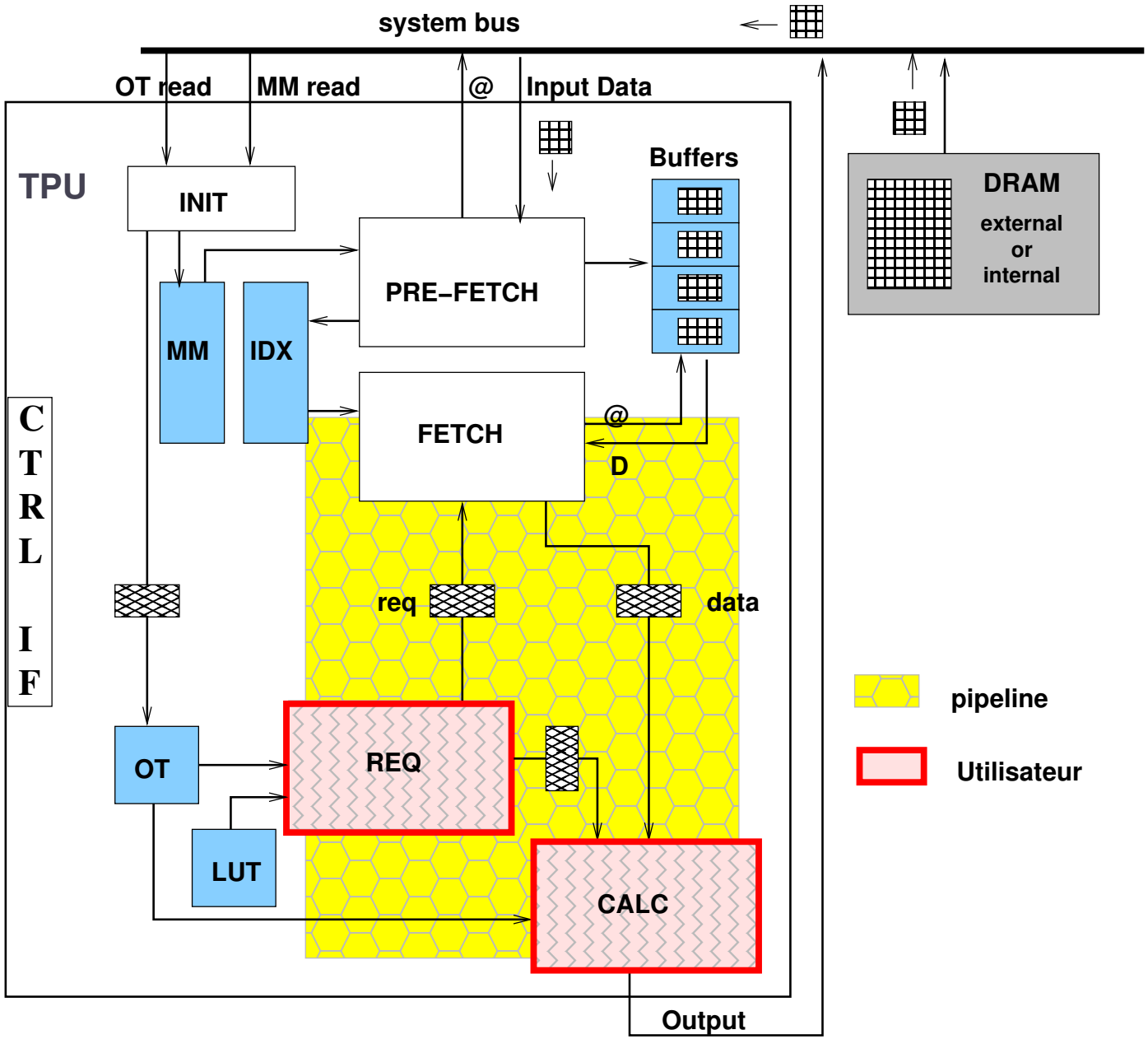
L'architecture cible: le TPU (Tile Processing Unit)



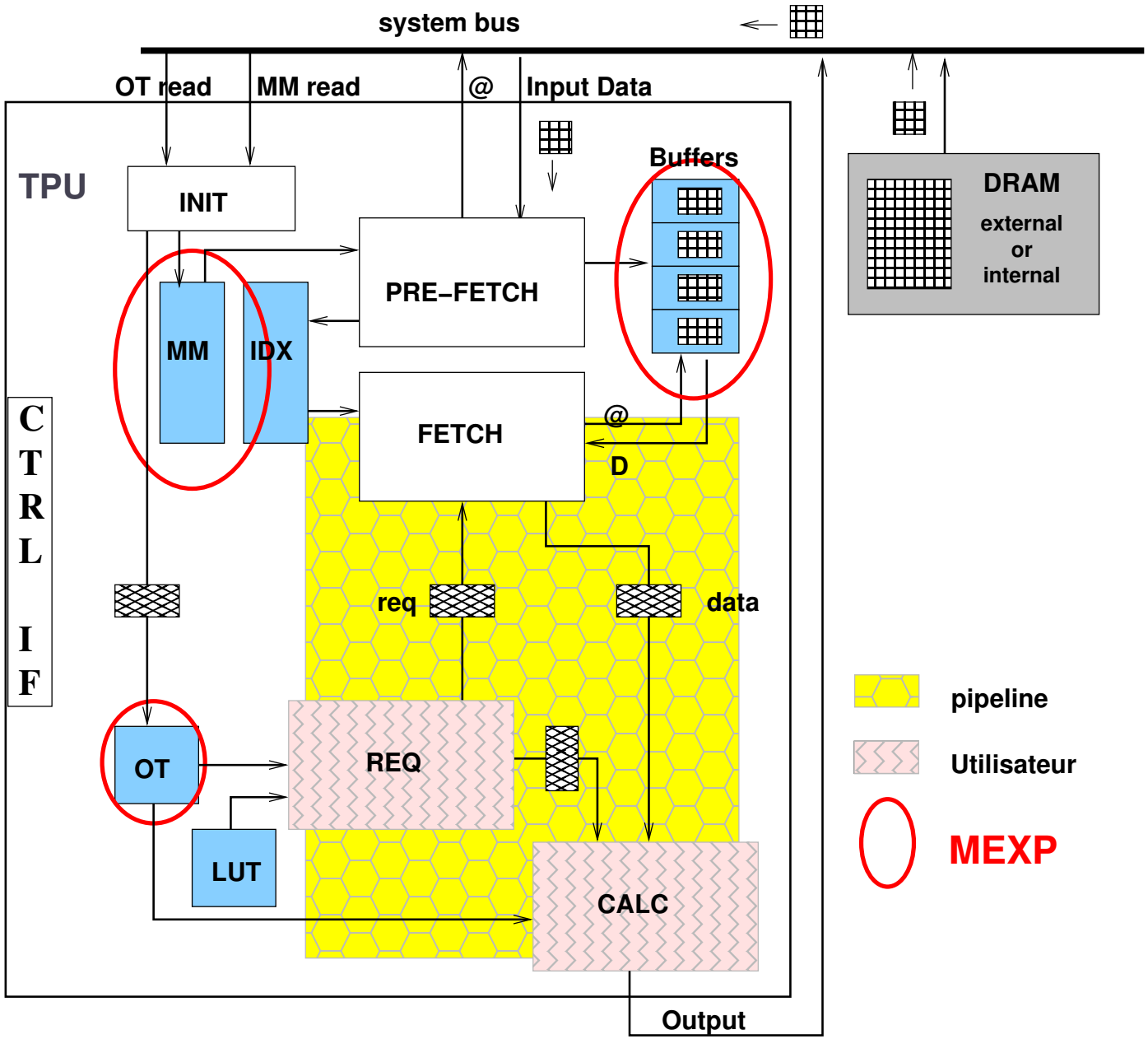
L'architecture cible: le TPU (Tile Processing Unit)



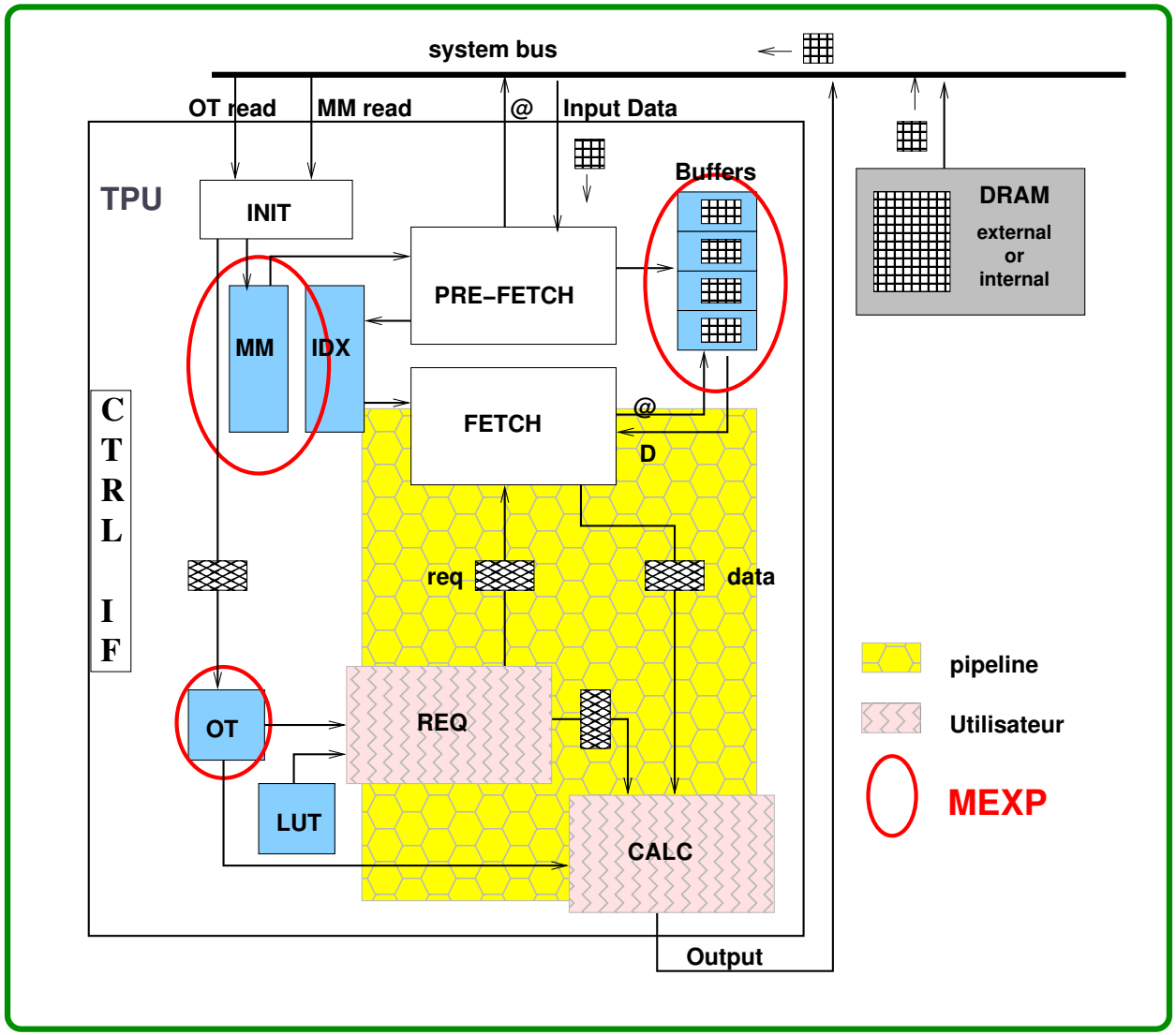
L'architecture cible: le TPU (Tile Processing Unit)



L'architecture cible: le TPU (Tile Processing Unit)

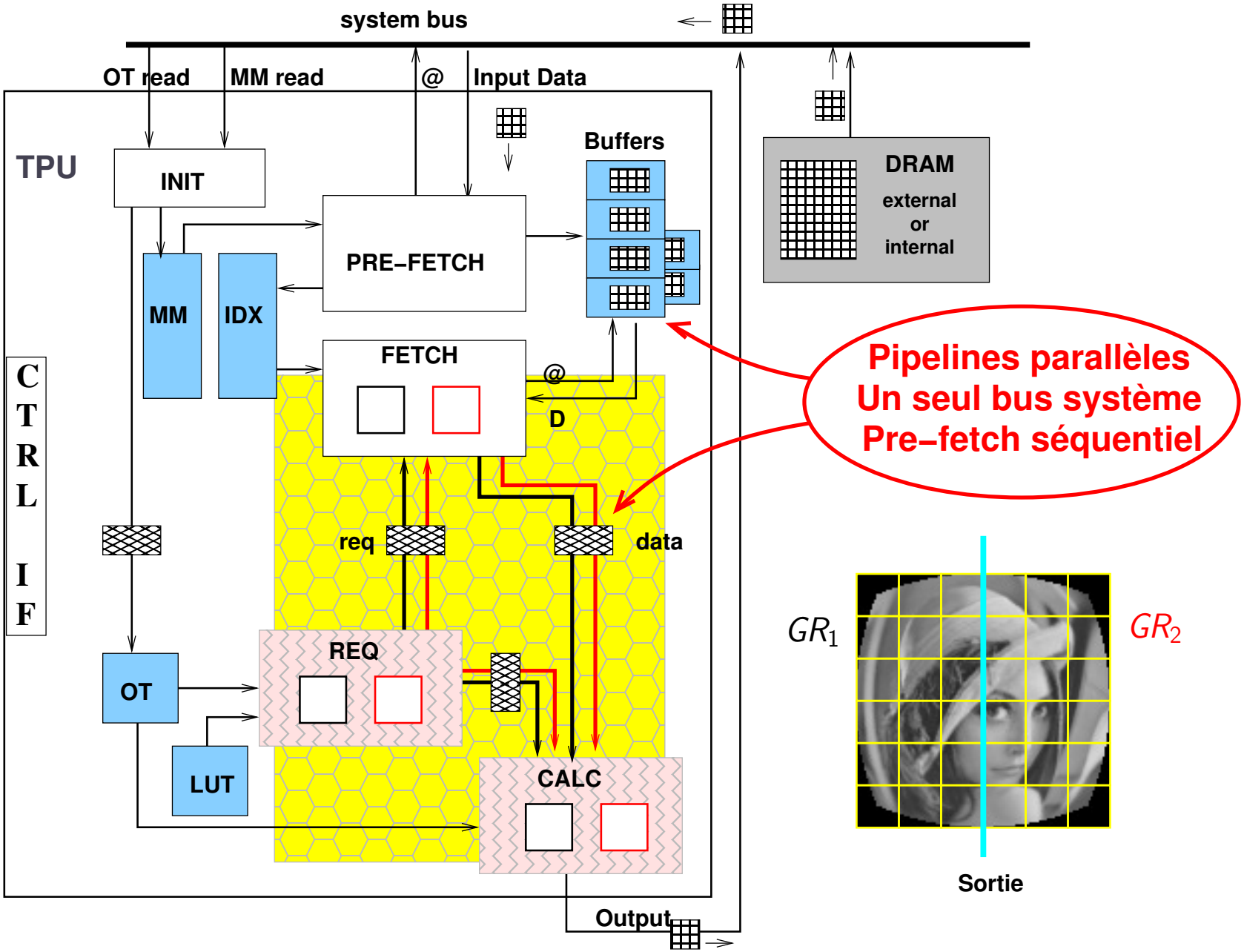


L'architecture cible: le TPU (Tile Processing Unit)

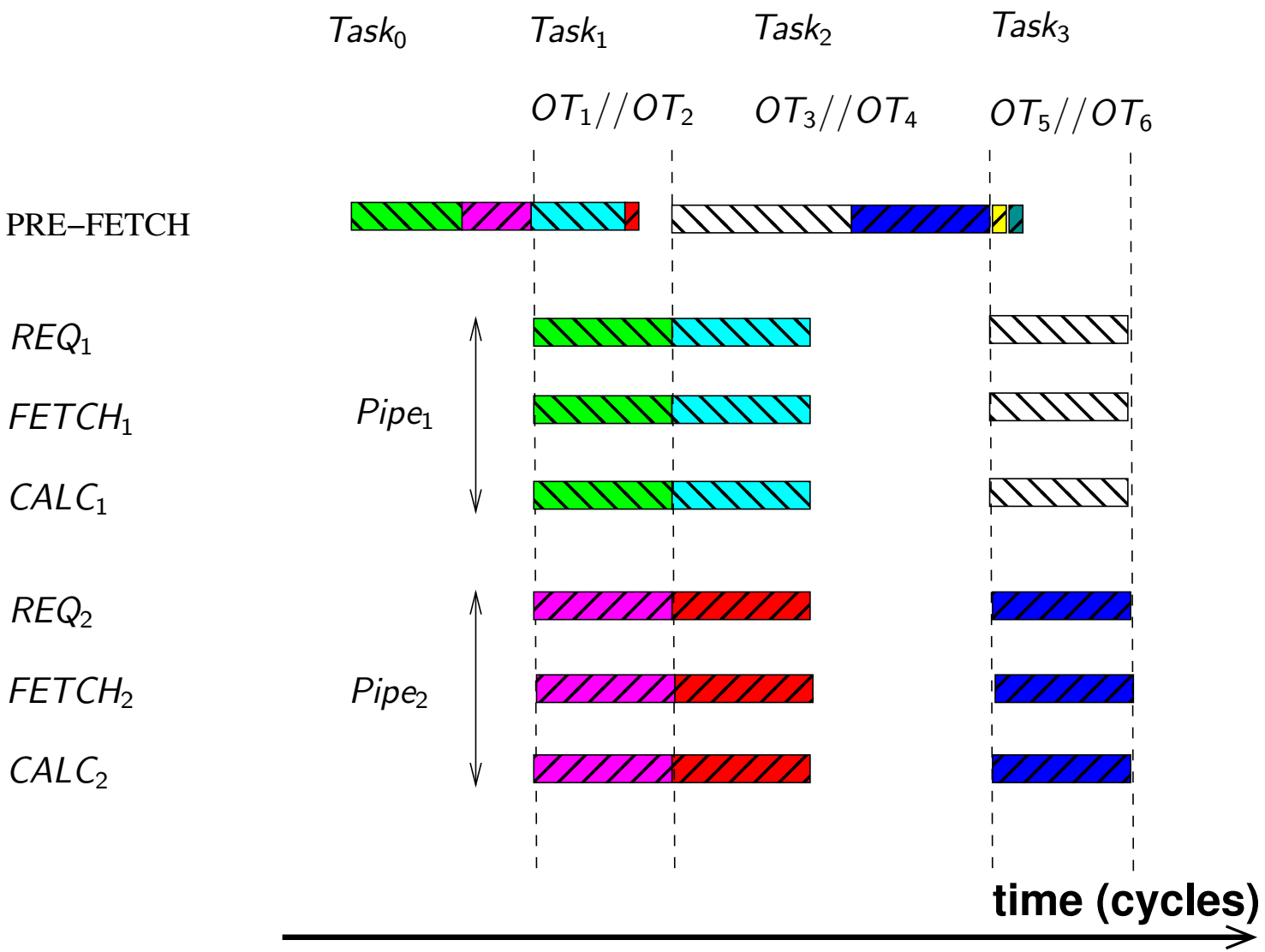


Modèle en C customisé

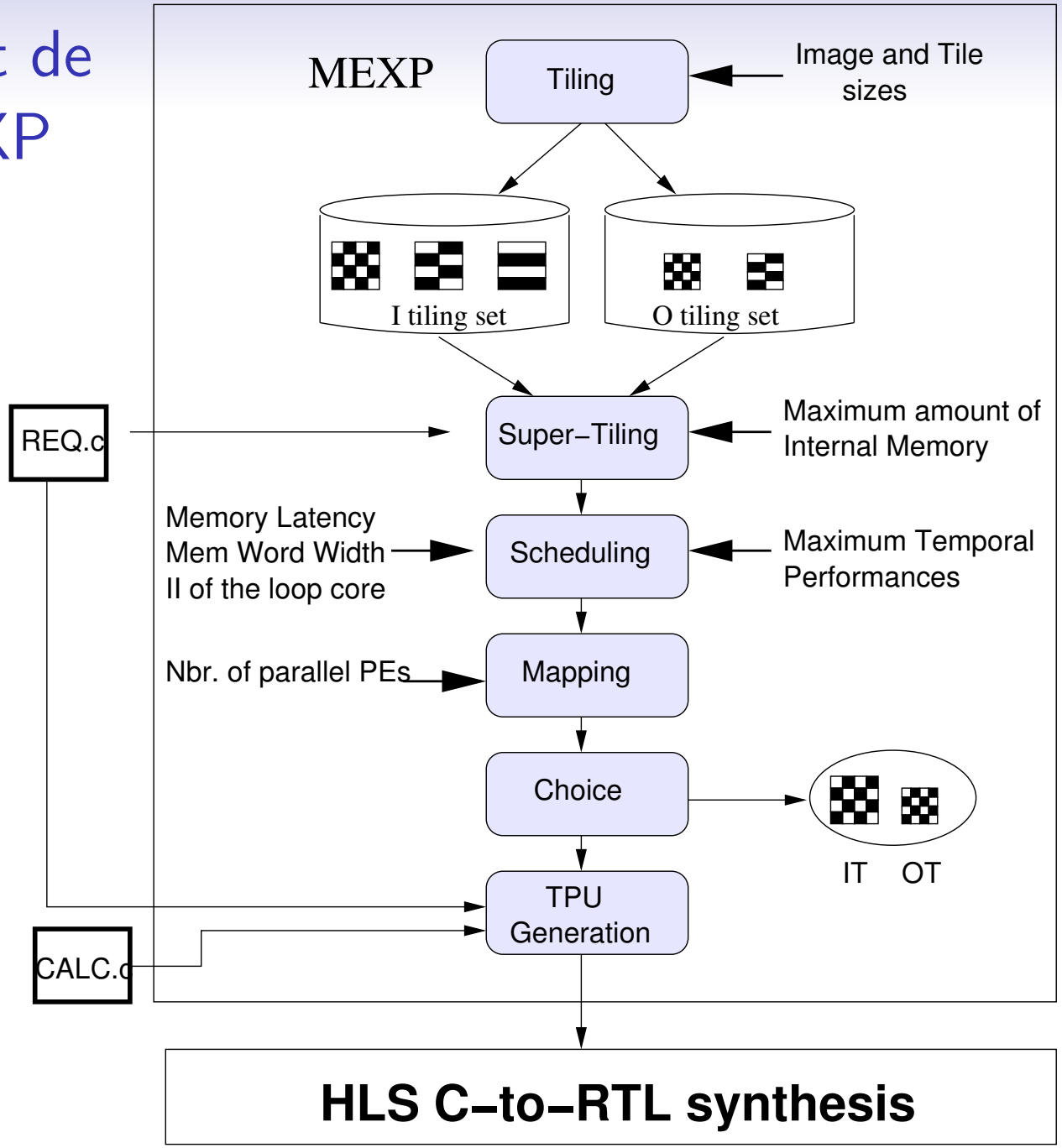
Le parallélisme inter-tuiles du TPU



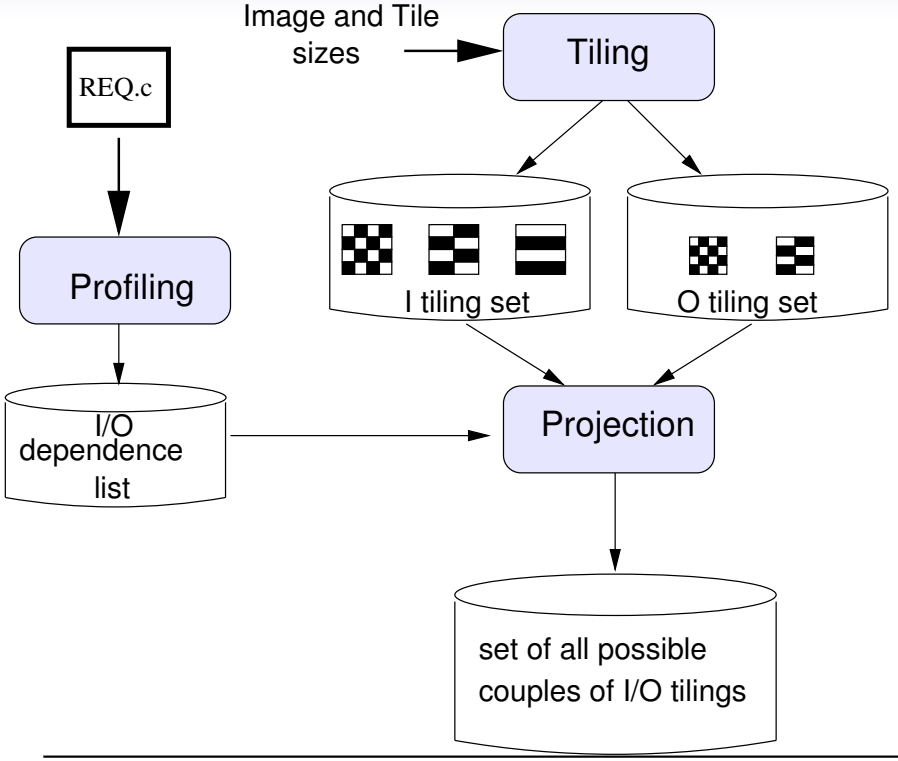
L'exécution temporelle d'un TPU avec deux pipelines parallèles



Le flot de MEXP

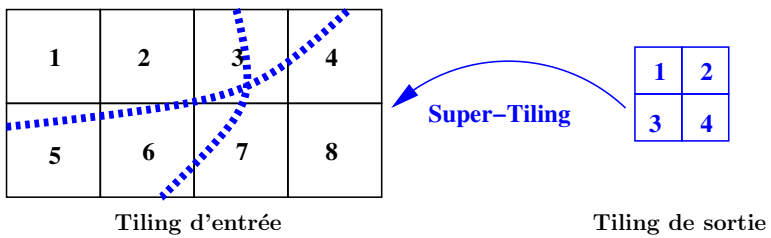


Le Super-Tiling et la matrice Σ décrivant une solution



- Exécution d'une fonction définie par l'utilisateur
- Calcul des dépendances d'entrée/ sortie
- Calcul des projections basé sur ces dépendances

Une solution est une couple de tilings d'entrée/sortie



Une matrice donne la correspondence entre OT et IT

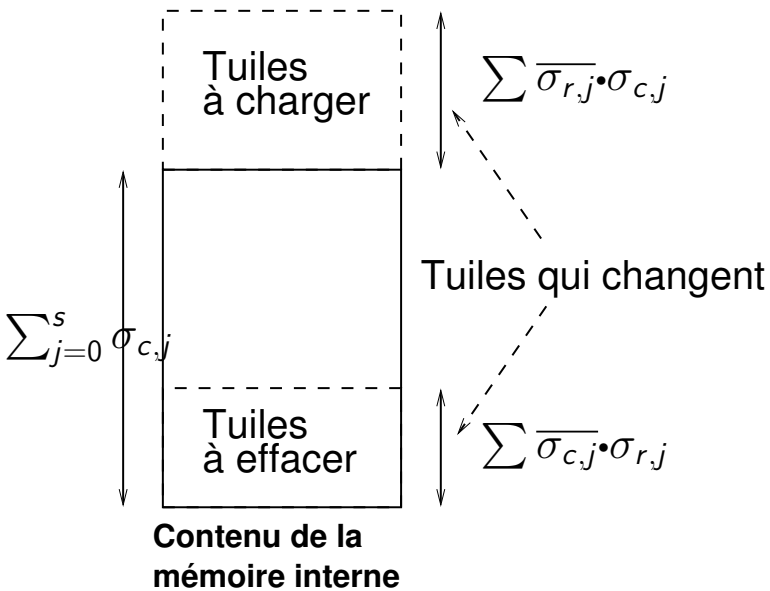
$$\Sigma = \underbrace{\begin{pmatrix} 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \end{pmatrix}}_{IT} \Big\} OT$$

Utilisation de la matrice Σ pour le calcul de critères

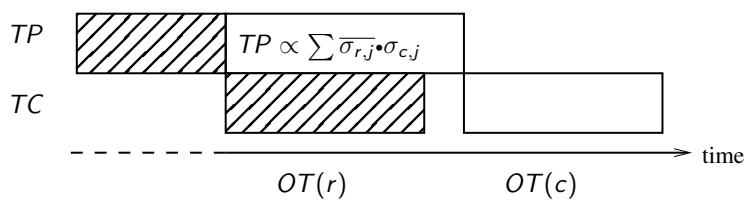
$$OT\ nbr. \left\{ \quad \Sigma^{4\times 8} = \underbrace{\begin{pmatrix} 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \end{pmatrix}}_{IT\ \text{nbr.}} \right. \quad \begin{matrix} \sum_j \sigma_{i,j} \\ 5 \\ 2 \\ 4 \\ 5 \end{matrix}$$

$$\max_i\left(\sum_j \sigma_{i,j}\right) = 5$$

Tuiles d'entrée dans la mémoire interne



Durée des temps de pre-fetch et de calcul



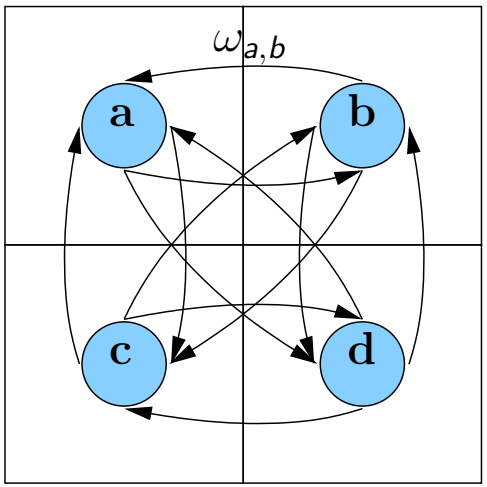
TC : Temps de calcul
TP : Temps de pre-fetch

Critères utilisés dans les étapes suivantes du flot

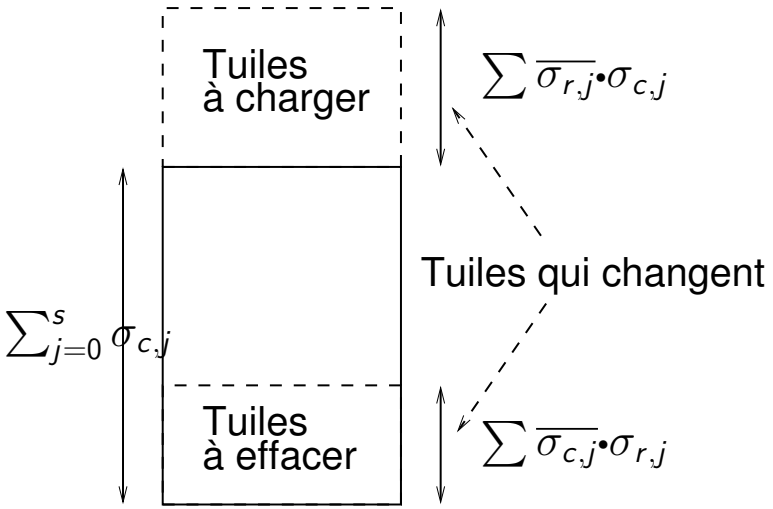
Le Scheduling (1)

- Ordonnancer les tuiles de sortie pour optimiser le matériel
- Associer un graphe au tiling de sortie
- Coût pour passer du calcul d'une tuile de sortie à une autre:

  Le nombre de tuiles d'entrée qui changent



Tiling de sortie



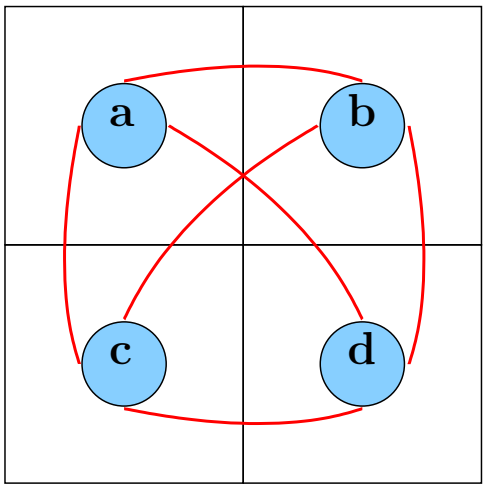
Contenu de la mémoire interne

Trouver un chemin passant par tous les nœuds une seule fois et minimisant un critère

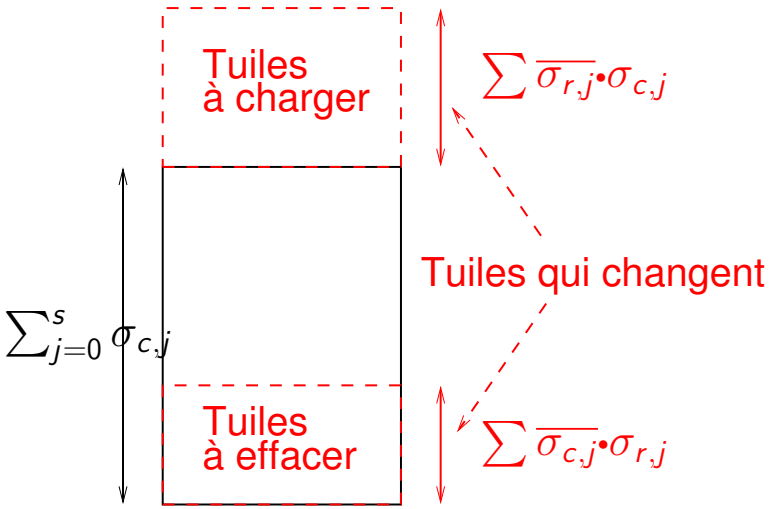
Le Scheduling (1)

- Ordonnancer les tuiles de sortie pour optimiser le matériel
- Associer un graphe au tiling de sortie
- Coût pour passer du calcul d'une tuile de sortie à une autre:

  Le nombre de tuiles d'entrée qui changent



Tiling de sortie



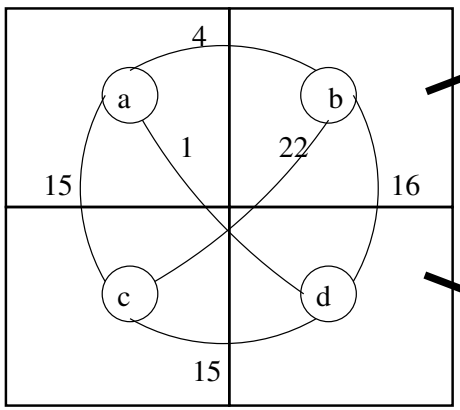
Contenu de la mémoire interne

Trouver un chemin passant par tous les nœuds une seule fois et minimisant un critère

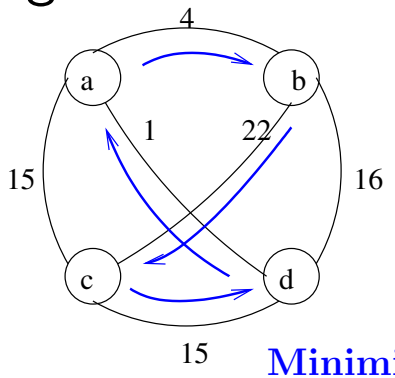
Le Scheduling (2)

Selon le critère de minimisation, résoudre:

1. Le Problème du Voyageur de Commerce (PCV) Classique
2. Le PVC qui minimise les goulots d'étranglement (EPVC)



Tiling de sortie

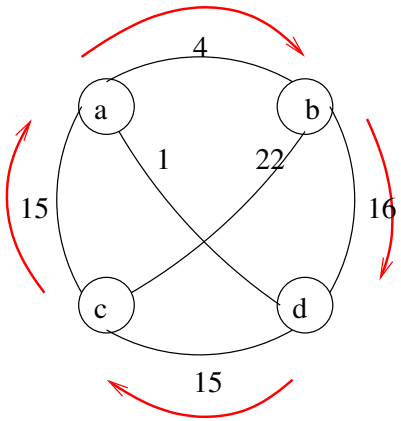


PVC : a, b, c, d

Cout Total = 42

Cout le plus élevé = 22

Minimiser le nbr. totale d'accès à la mémoire externe



EPVC : a, b, d, c

Cout Total = 50

Cout le plus élevé = 16

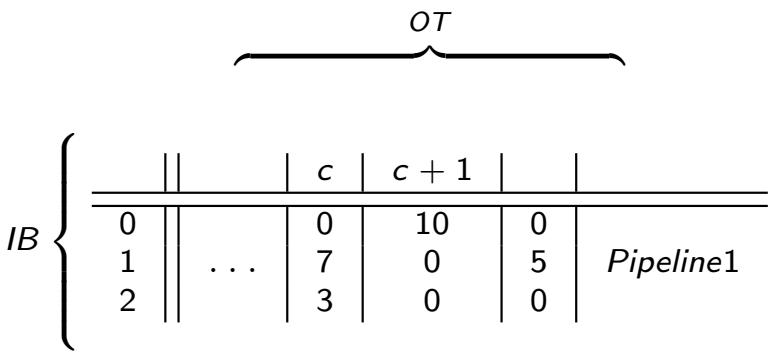
Minimiser le temps de pre-fetch max

Résolution par algorithme génétique

Le Mapping

- Distribution des calculs des différents groupes aux pipelines parallèles
- Allocation du nombre de buffers nécessaires à chaque pipeline
- Placement des tuiles d'entrée dans les différents buffers mémoire

Mémoire de Mapping pour un pipeline:

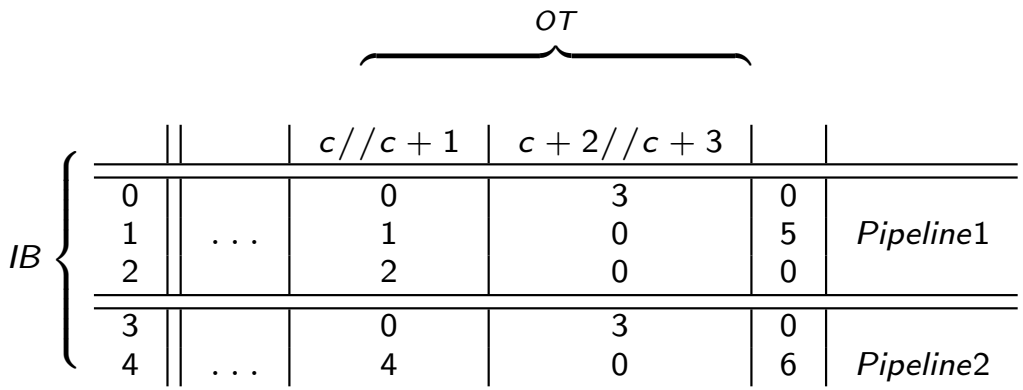


Lorsque nous calculons la tuile de sortie **c**, la tuile d'entrée 7 est dans le buffer 1

Le Mapping

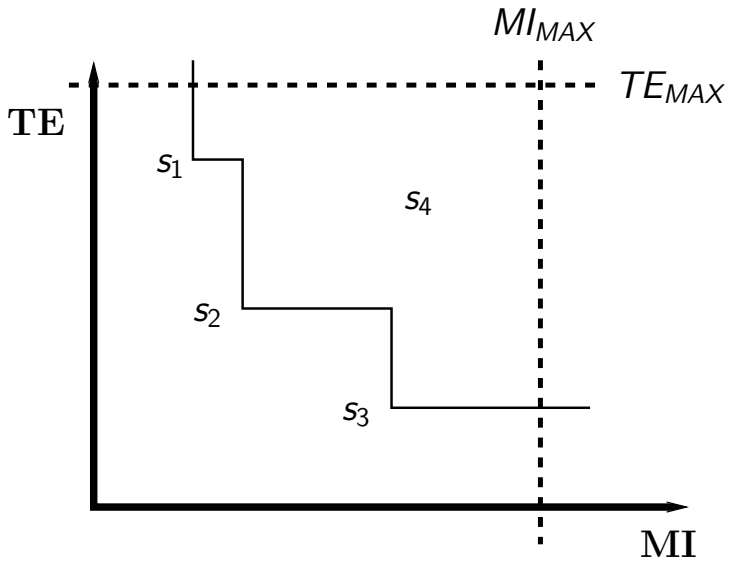
- Distribution des calculs des différents groupes aux pipelines parallèles
- Allocation du nombre de buffers nécessaires à chaque pipeline
- Placement des tuiles d'entrée dans les différents buffers mémoire

Mémoire de Mapping pour 2 pipelines parallèles:



Des groups de buffers indépendants sont réservés aux pipelines parallèles

Le Choix des solutions



- Deux critères de selection calculés de la matrice Σ:
  - Mémoire interne (MI)
  - Temps d'Exécution (TE) en nombre de cycles
- Les solutions choisies sont Pareto:
  - utilisant le minimum de MI pour un TE donné
  - ayant le TE minimum pour une MI donnée

Conclusion

1. MEXP explore des couples possibles de tilings pour l'entrée et la sortie
2. Pour chaque couple
   - calcule une matrice de correspondances entre les tuiles d'entrée et de sortie
   - trouve un ordonnancement optimisé des calculs des tuiles de sortie
   - évalue la quantité de mémoire interne utilisée et le temps d'exécution de l'algorithme sur le matériel cible
3. Sur l'ensemble des couples, MEXP choisit les solutions Pareto
4. Pour une solution Pareto choisie par l'utilisateur, MEXP génère un programme en C-synthétisable en customisant un programme en C générique

Plan

Conditions expérimentales

MEXP a été testé pour trois algorithmes:

- L' **échantillonnage logarithmique**
- La **transformation polaire**
- Le **filtrage pyramidal**

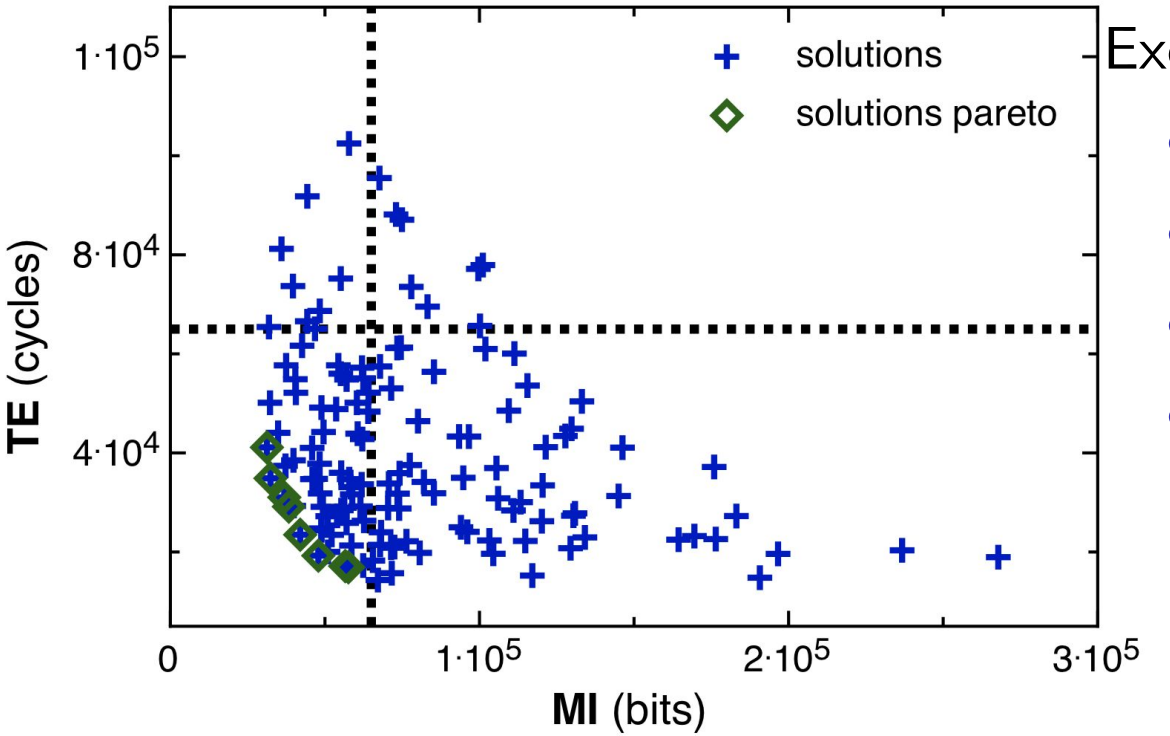
Algorithmes analysés avec:

- **Tailles d'image** d'entrée: SQCIF, VGA, HDTV
- **Niveaux de parallélisme**: N_p=4, 2, 1
- **Latences mémoire**: LAT=100, 60, 30

Caractéristiques des explorations faites:

- Temps d'execution d'une exploration $\in$ [qq. sec., qq. h]
- Nbr. de solutions dans l'espace d'explorations $\in$ [36, 324]
- Nbr. tuiles d'entrée par solution $\in$ [128, 8192]
- Nbr. de tuiles de sortie par solution $\in$ [64, 2048]

Exemple d'une exploration de MEXP

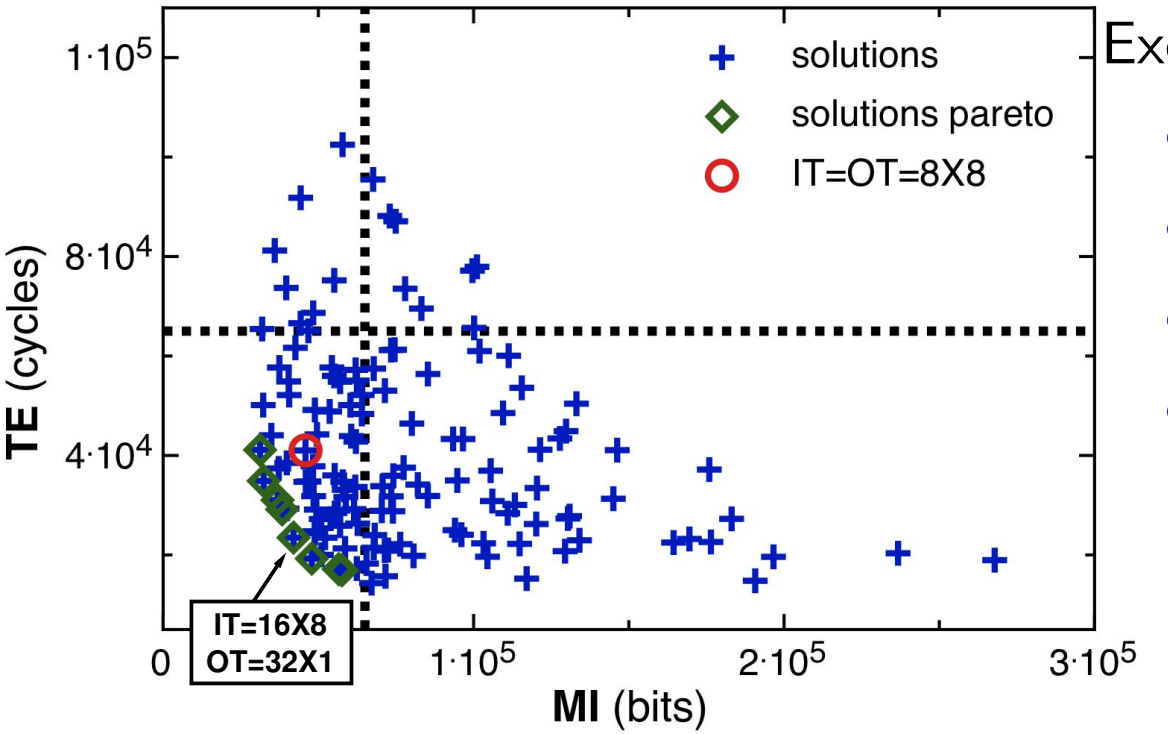


Exemple d'exploration:

- Transformation Polaire
- Latence mémoire 100 cycles
- Pipelines parallèles N_p=4
- Taille d'image d'entrée SQCIF

- espace avec 132 solutions
- différences importantes entre solutions
- solution choisies non-triviales

Exemple d'une exploration de MEXP



Exemple d'exploration:

- Transformation Polaire
- Latence mémoire 100 cycles
- Pipelines parallèles N_p=4
- Taille d'image d'entrée SQCIF

- espace avec 132 solutions
- différences importantes entre solutions
- solution choisies non-triviales

Caractéristiques des solutions après synthèse

Les solutions Pareto issues de l'exploration de MEXP sont transmises à l'outil de Synthèse de Haut niveau pour être synthètisées

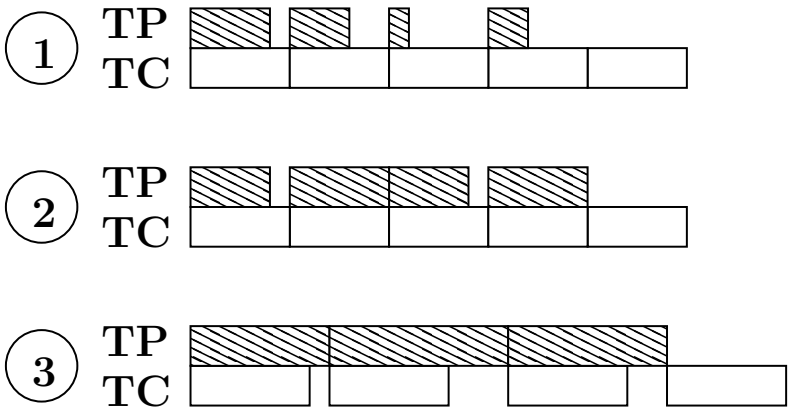
Pour chaque solution nous évaluons

- les temps d'exécution de l'algorithme
- la surface occupée

valeurs évaluées par simulation logique après synthèse réalisée sous PICO (Synfora)

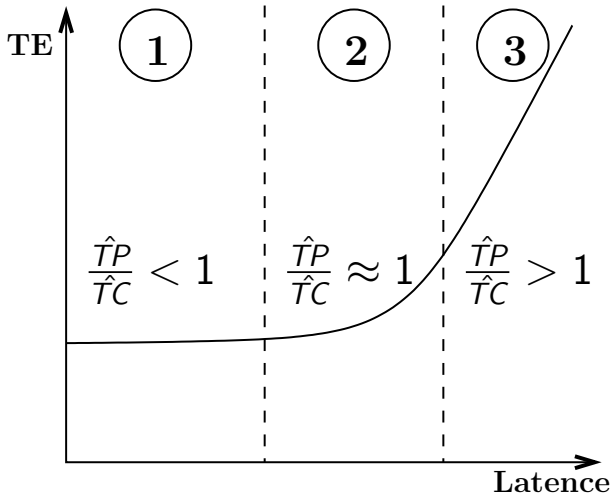
Temps d'exécution des solutions (1)

Le temps d'exécution (TE) d'une solution dépend du rapport entre le temps de pre-fetch (TP) et le temps de calcul (TC)



Le temps de pre-fetch dépend de:

- Lois d'accès de l'algorithme cible
- Taille des tuiles et des images
- Niveau de parallélisme
- Latence mémoire



$\hat{TC}$ **temps de calcul moyen**

$\hat{TP}$ **temps de pre-fetch moyen**

Temps d'exécution des solutions (2)

Ex. transformation polaire

Image d'entrée 128X128 pixels

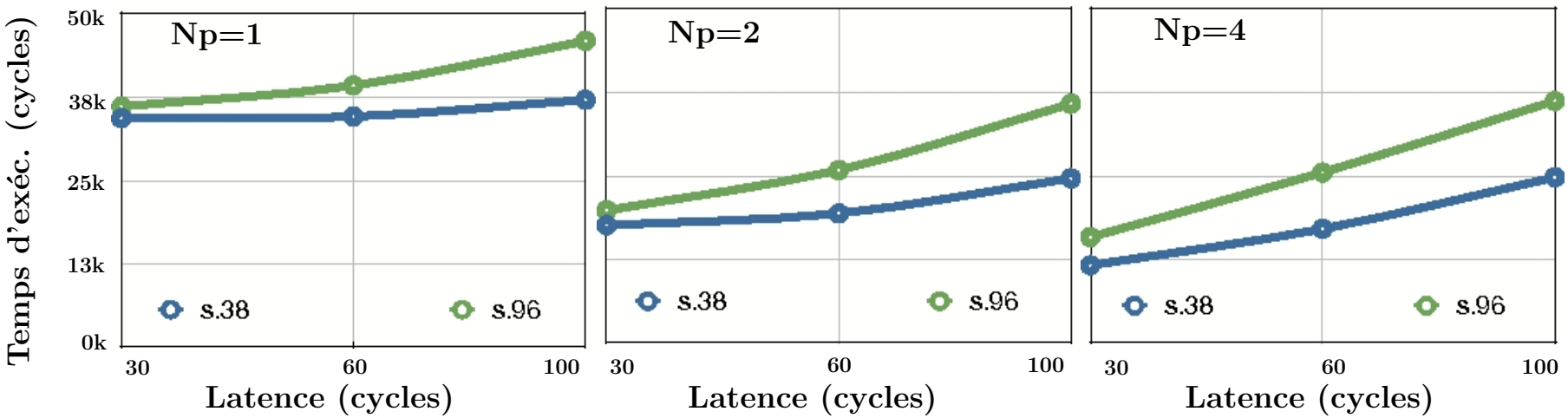
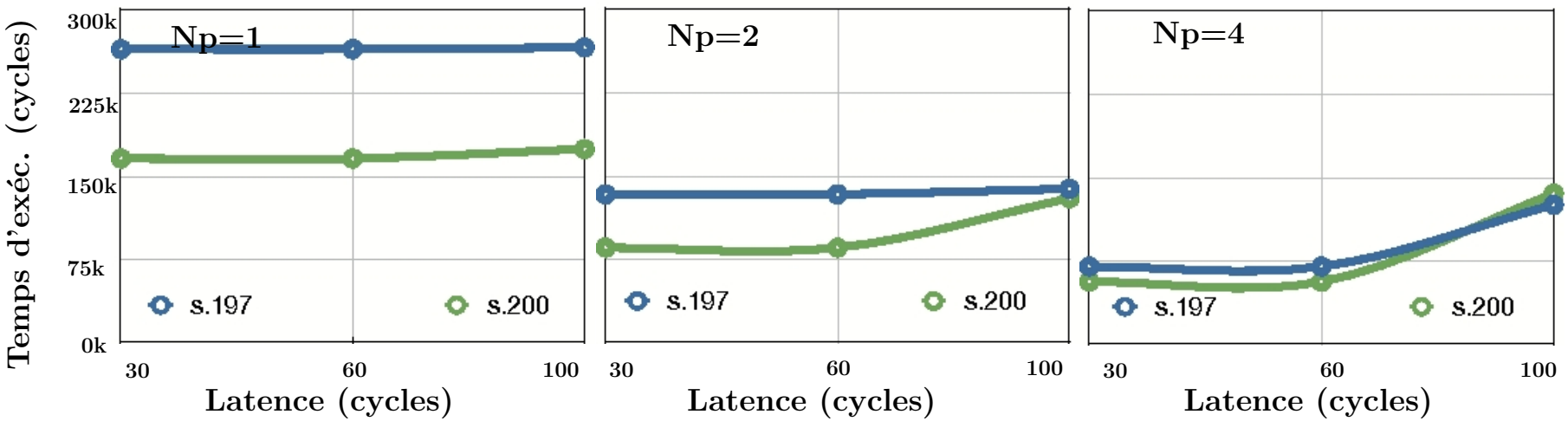


Image d'entrée 600X600 pixels



Temps d'exécution des solutions (2)

Ex. transformation polaire

Image d'entrée 128X128 pixels

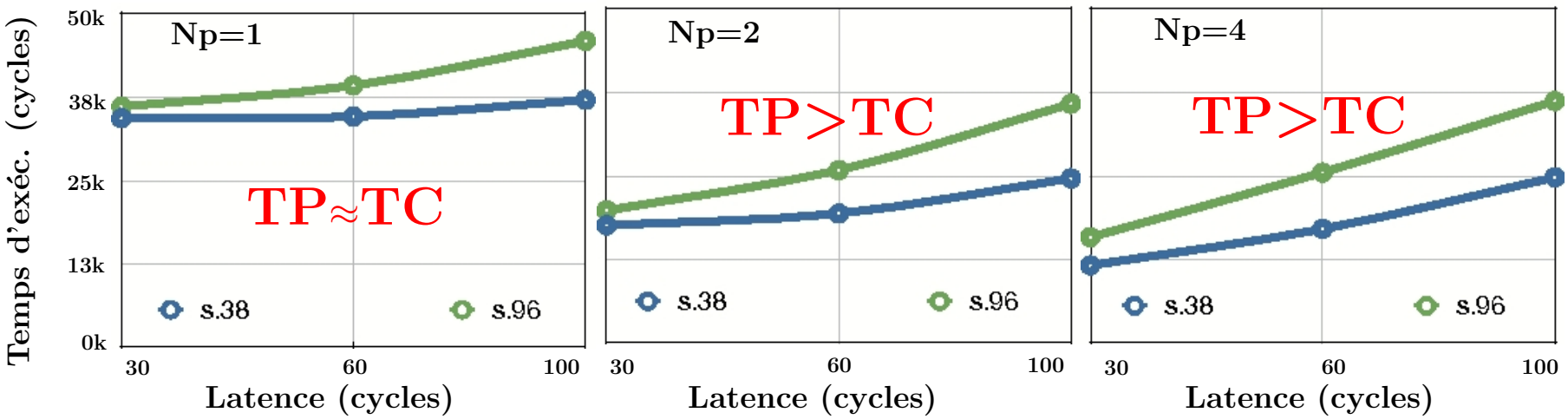
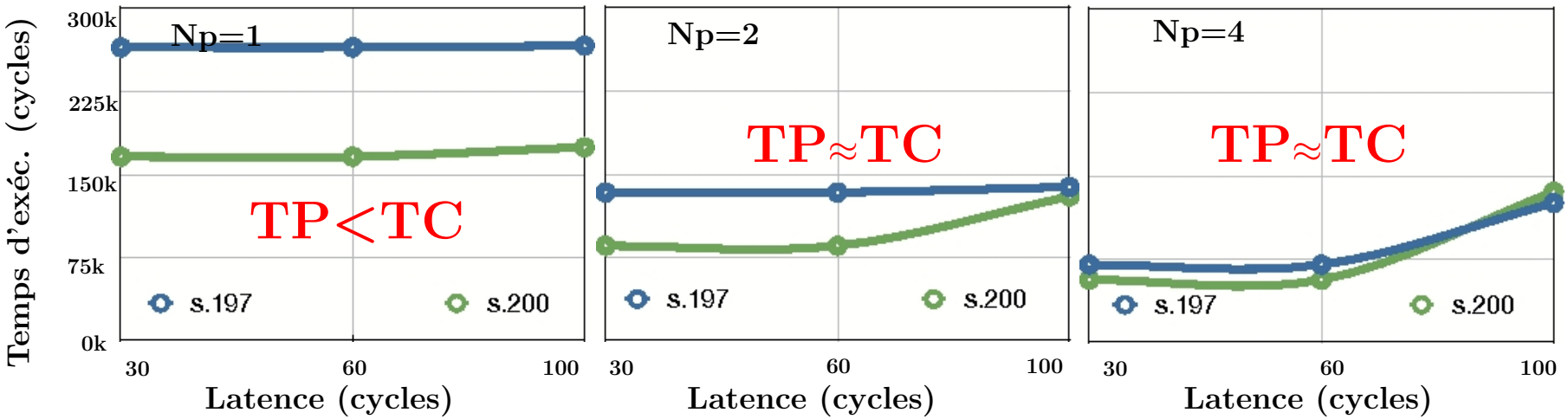


Image d'entrée 600X600 pixels



Temps d'exécution des solutions (2)

Ex. transformation polaire

Image d'entrée 128X128 pixels

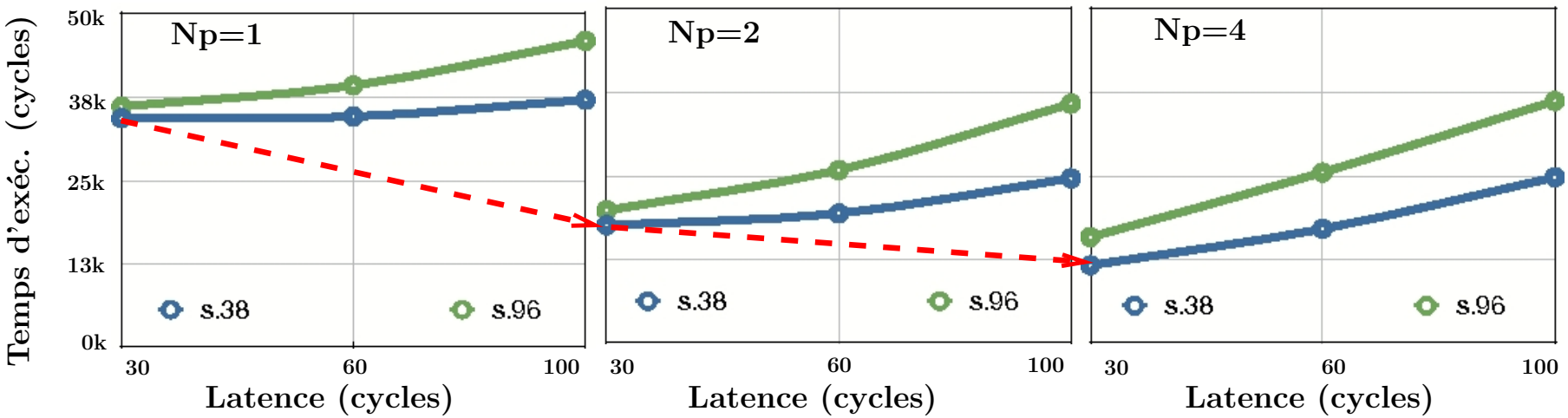
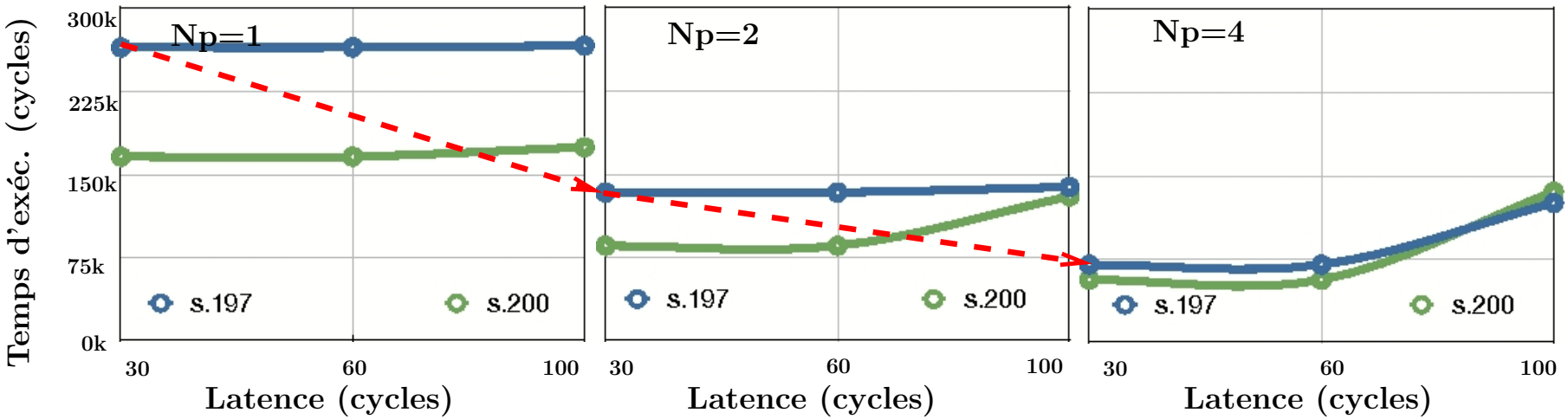


Image d'entrée 600X600 pixels



Temps d'exécution des solutions (2)

Ex. transformation polaire

Image d'entrée 128X128 pixels

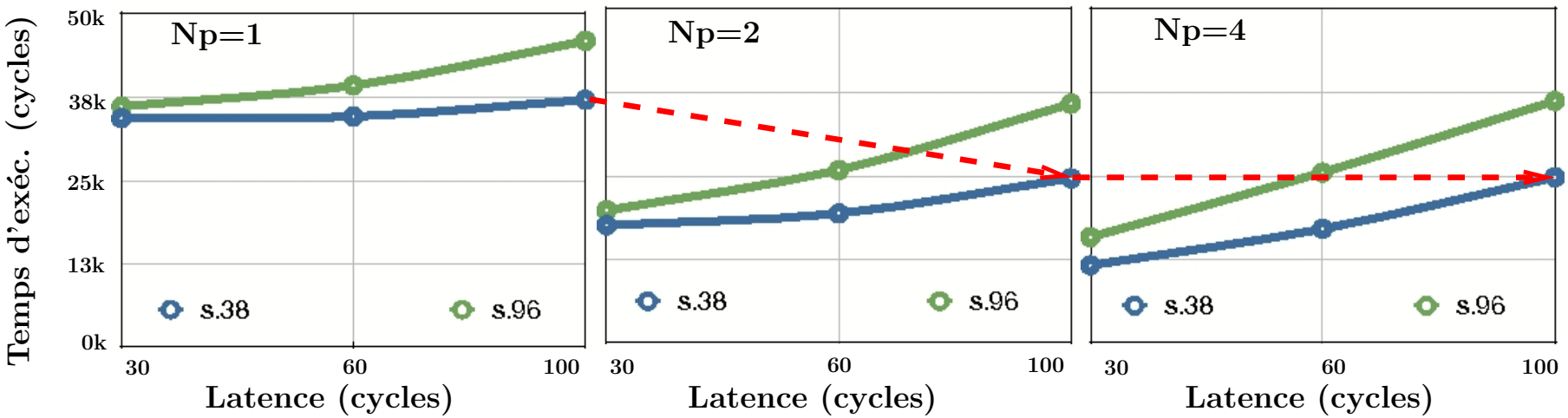
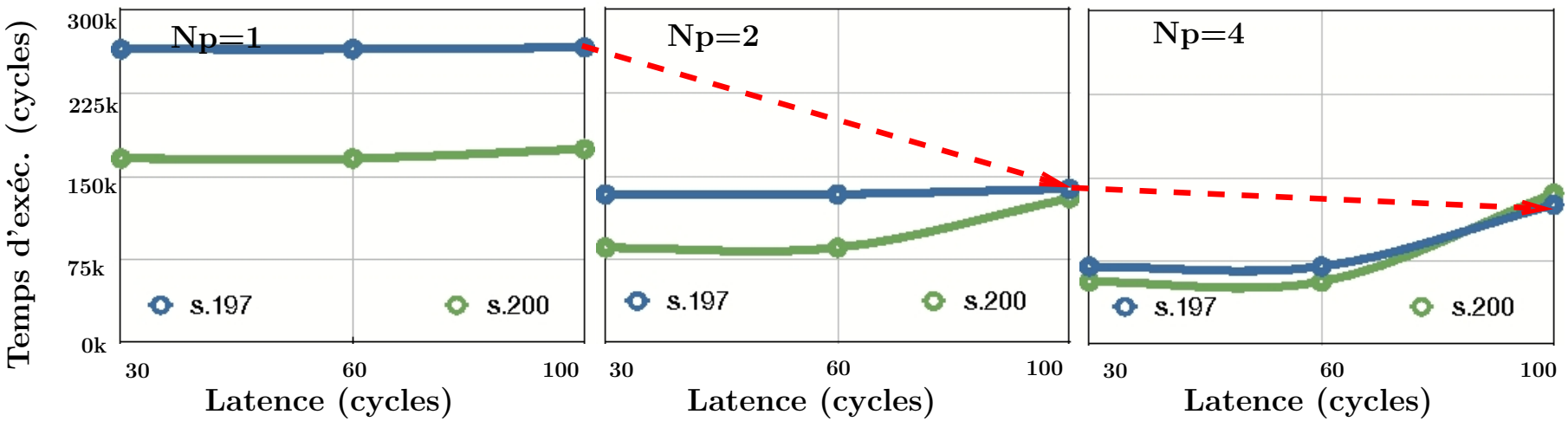


Image d'entrée 600X600 pixels

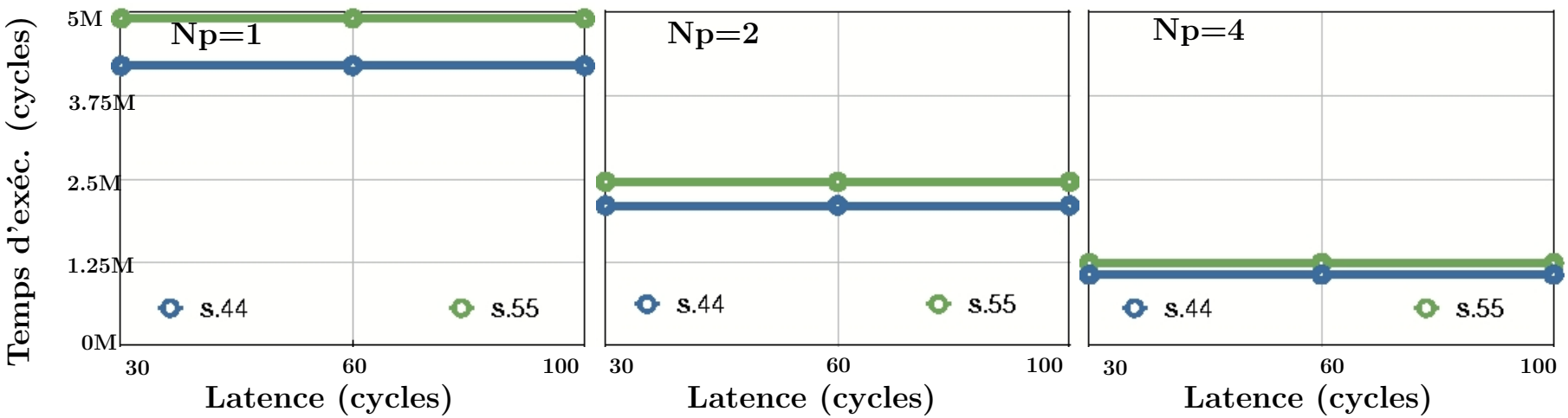


Temps d'exécution des solutions(3)

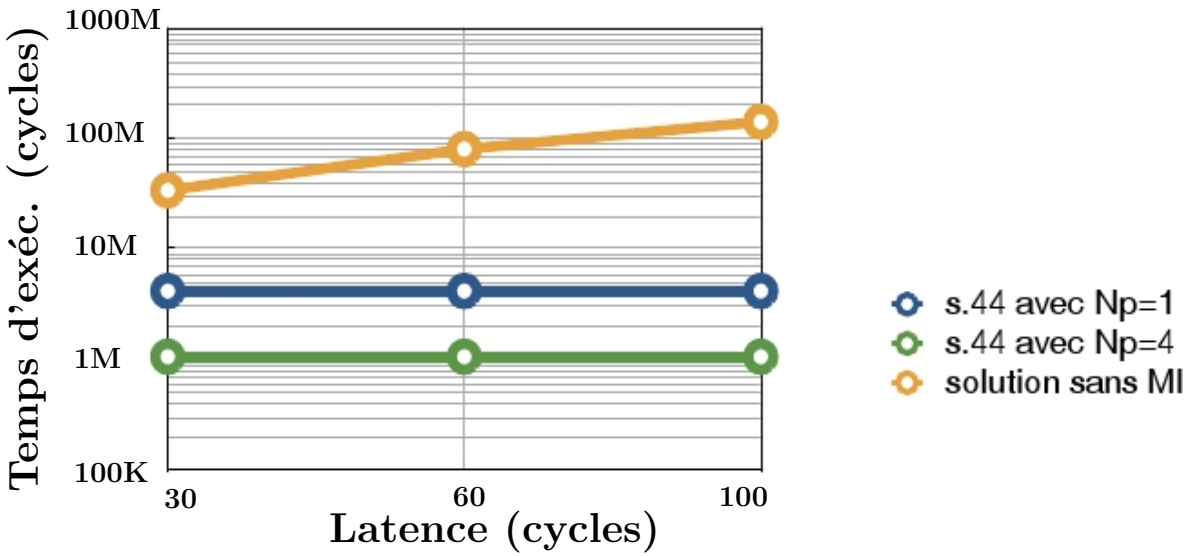
Ex. filtre Pyramidal

Choix d'un couple de tiling d'entrée et de sortie qui permet de masquer le temps de pre-fetch par le temps de calcul

Parallélisme efficace



Accélérations par rapport à une solution sans mémoire interne



- Accélération maximale sans parallélisme **34X**
  ⇒ augmentation de surface **6.9X**
- Accélération maximale avec parallélisme **134X**
  ⇒ augmentation de surface **21.9X**

Résultats obtenus pour l'échantillonnage pyramidal sur HDTV

Facteurs moyens d'augmentation de la surface

par rapport à une solution sans mémoire interne

	$N_p = 1$			...	$N_p = 4$		
	tot.	combi.	seq. +M	...	tot.	combi.	seq.+M
SQCIF	2.3	1.8	4	...	8.1	4.6	12.3
VGA	5.5	1.9	17.1	...	16.9	5.3	56.7
HDTV	7.9	2	26.5	...	25.1	5	87.5

- La surface dédiée à la logique combinatoire (combi.)
  varie peu avec la taille de l'image d'entrée
  augmente avec le niveau de parallélisme

- La surface dédiée à la logique séquentielle (seq.) + mémoires
  augmente avec la taille de l'image d'entrée
  augmente avec le niveau de parallélisme

Taille de la mémoire interne $\approx 1\ mm^2$

Cas de la transformation polaire (HDTV):
Augmentation de surface mémoire la plus élevée: 116X.

sans mémoire interne				MI et $N_p = 4$		
	mm^2				mm^2	
tot.	(comb	seq.)	. . .	tot.	(comb	seq.+M)
0.03	(0.02	0.01)		1.050	(0.144	**0.9**)

Cas du filtrage Pyramidale (HDTV):
Accéleration du temps d'exécution la plus élevée: 134X.

sans mémoire interne				MI et $N_p = 4$		
	mm^2				mm^2	
tot.	(comb	seq.+M)	. . .	tot.	(comb	seq.)
0.0390	(0.030	0.009)		1.50	(0.2	**1.3**)

Efficacité du parallélisme

Facteurs dûs au parallélisme et évalués par rapport à une solution utilisant un seul pipeline

AS= Augmentation de la surface

ATE= Accélération du temps

$$\mathbf{E} = \frac{\mathbf{ATE}}{N_p}$$

$$0 \leq \mathbf{E} \leq 1$$

$\mathbf{E} > 0.5 \Rightarrow$ parallélisme efficace

	$\hat{\mathbf{E}}$	$\mathbf{E}_{max}$	$\hat{\mathbf{AS}}$
Np=2	0.8	0.99	1.6
Np=4	0.7	0.99	2.9

Il est possible de trouver des couple de tiling d'entrée et de sortie qui maximisent l'efficaciité du parallélisme

Conclusion Évaluation de MEXP

MEXP: 81 explorations

1 exploration = qqs. centaines de solutions

on retient 2 solutions Pareto par exploration

PICO:

synthèse de 162 solutions

évaluation par simulation logique

Résultats:

accélération maximale: 134X (augmentation de surface: 22X)

surface maximale : 1,3 mm^2

L'explorations de MEXP sont basées sur des estimations

Écart moyen entre les estimations et les valeurs mesurées après synthèse $<10\%$

Les explorations de MEXP sont fiables avec une amélioration significative des performances

Plan

Conclusion

- MEXP est un outil d' exploration d'espace des architectures pour un système de traitement d'image
- Peut être utilisé comme front-end de la synthèse de haut niveau
- Permet un compromis entre la taille de mémoire interne et les performances temporelles du système
- Permet un parallélisme inter-tuiles
- Les explorations de MEXP sont fiables avec une amélioration significative des performances

Perspectives

- Utilisaton de tuiles avec côtés non orthogonaux
- Étude des améliorations apportées par MEXP en terme de puissance consommée
- Adaptation de MEXP aux algorithmes récursifs ou aux algorithmes multi-étapes
- Utilisation de MEXP comme front-end des compilateur des multiprocesseurs programmables comme les GPU

Remerciments

- Jeanny Hérault (Directeur de thèse)
- Stéphane Mancini (Encadrant)
- Roberto Guizzetti (Encadrant)
- Dominique Houzet
- Pascal Urard
- Barthélémy Durette
- Florian Collard
- Christophe Planat
- Jean-Luc Bui
- Marc Ettori

Merci pour votre attention

DEMO