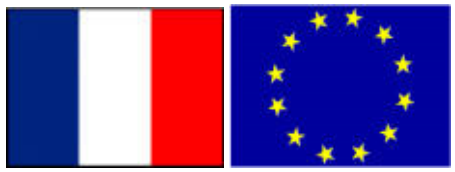
CentraleSupélec



GP-GPU

TP-3
CUDA programming with *Shared Memory*

Laercio LimaPilla & Stéphane Vialle

CentraleSupélec université PARIS-SACLAY ÉCOLE DOCTORALE Sciences et technologies de l'information et de la communication (STIC) RISEGrid Research Institute for Smarter Electric Grids région lorraine Grand Est ALSACE CHAMPAGNE-ARDENNE LORRAINE

Stephane.Vialle@centralesupelec.fr
http://www.metz.supelec.fr/~vialle

1 – No boundaries problem

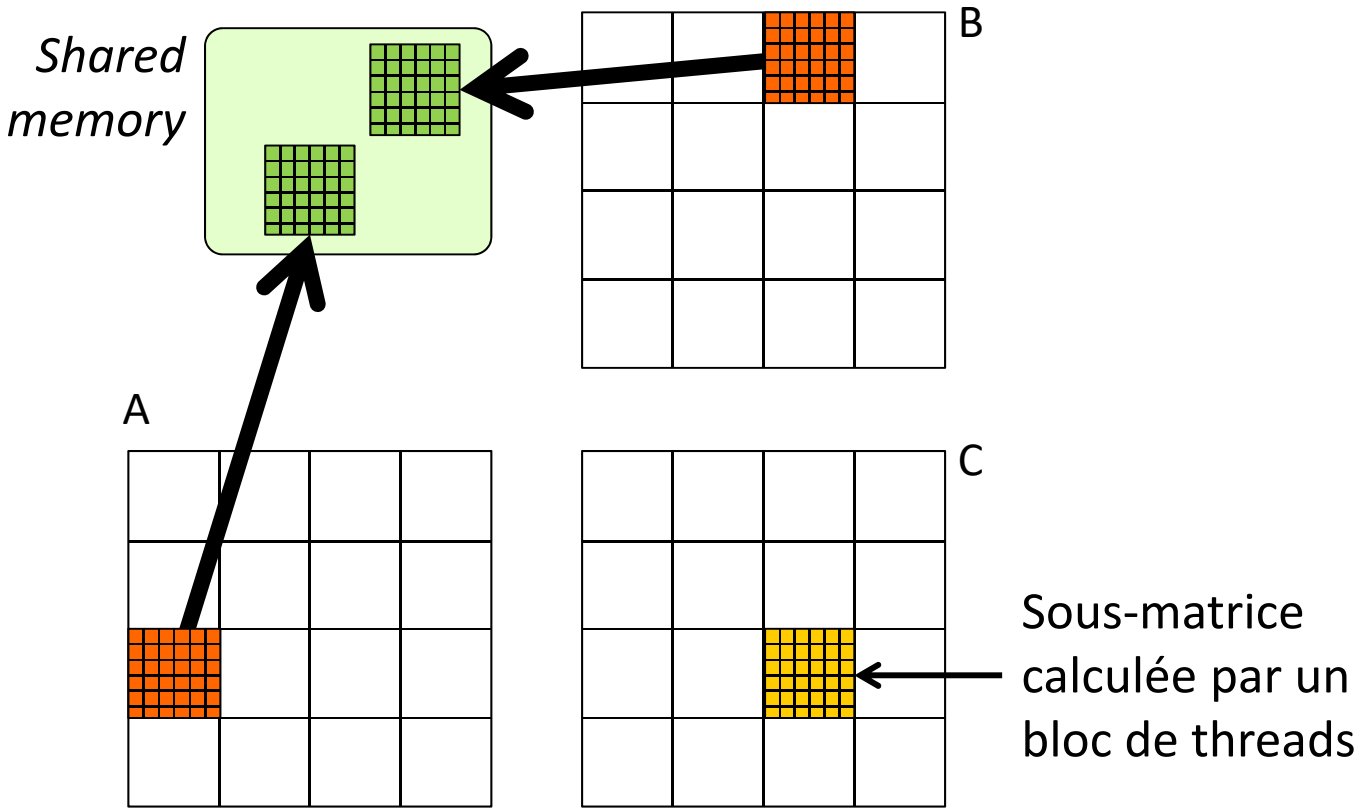
Blocs 2D de threads (*kernel k2*)

MatrixSide = *k.BlockSize_xy*

Step 0.a

Chargement de sous-matrices dans la *shared memory.*

Accès coalescents à la mémoire globale.

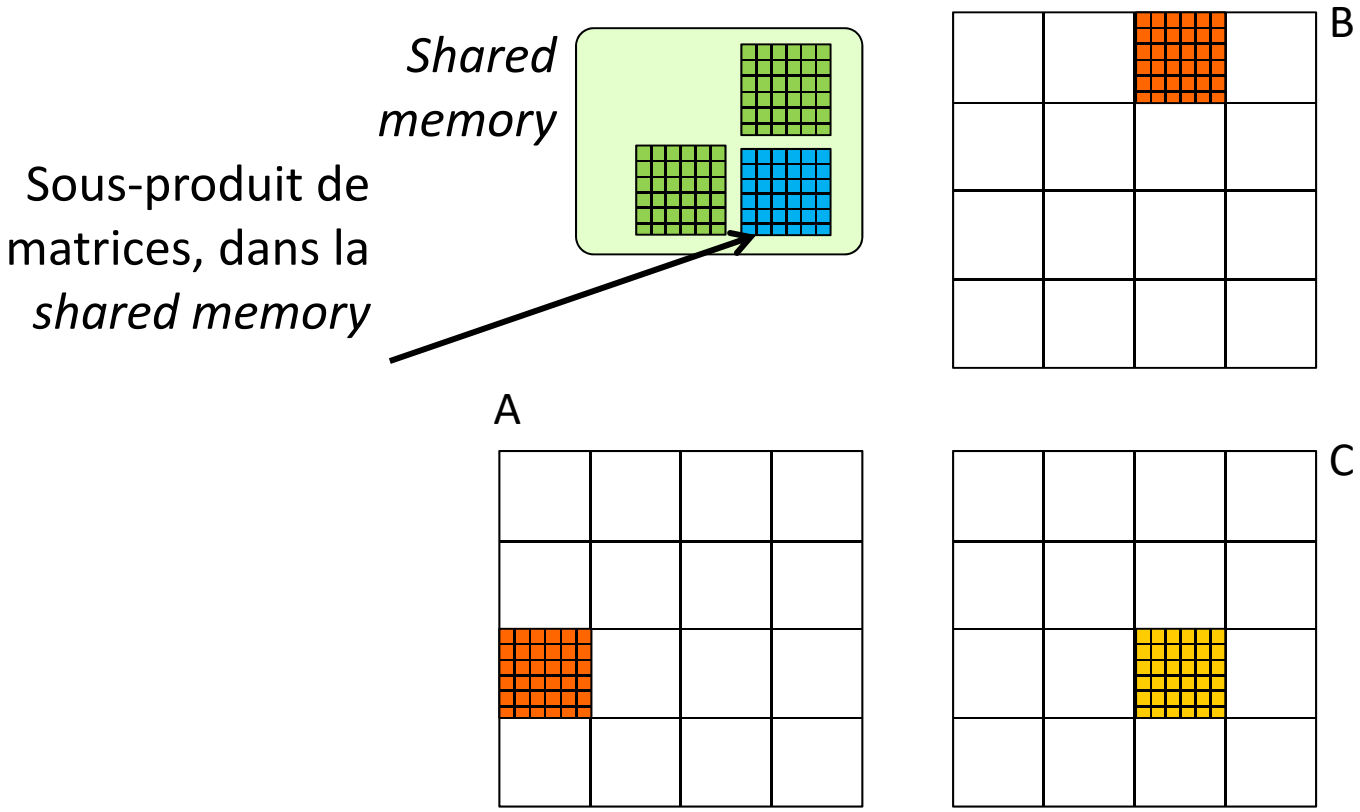


1 – No boundaries problem

Blocs 2D de threads (*kernel k2*)

MatrixSide = *k.BlockSize_xy*

Step 0.b



1 – No boundaries problem

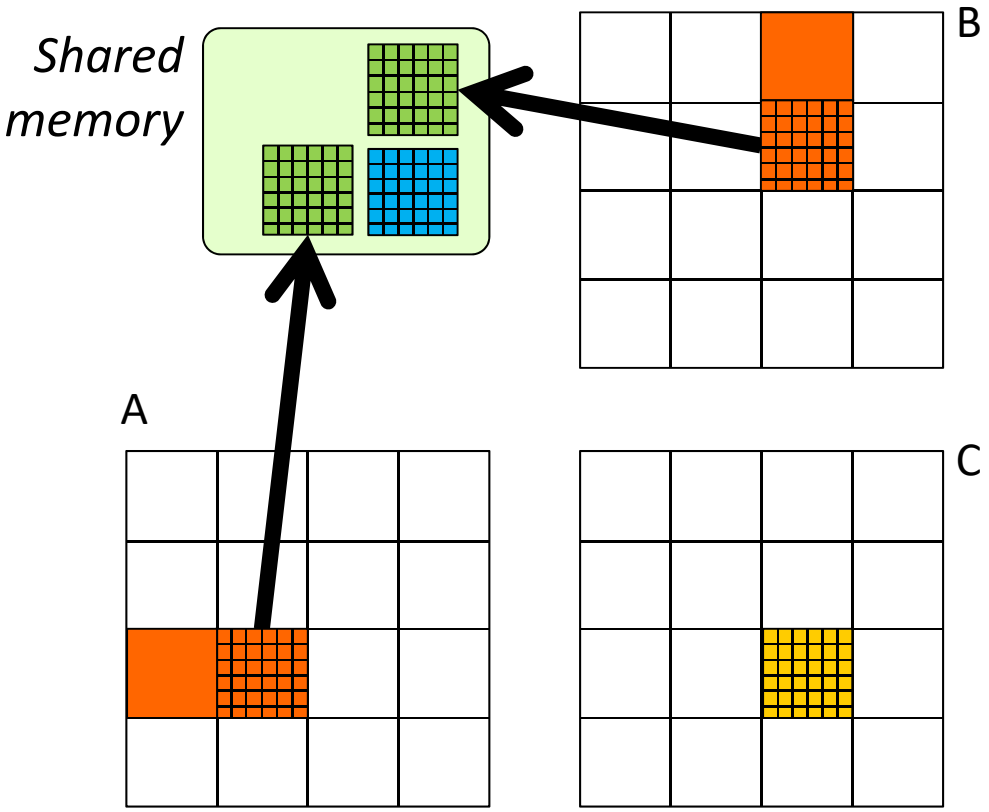
Blocs 2D de threads (*kernel k2*)

MatrixSide = *k.BlockSize_xy*

Step 1.a

Chargement de sous-matrices dans la *shared memory.*

Accès coalescents à la mémoire globale.

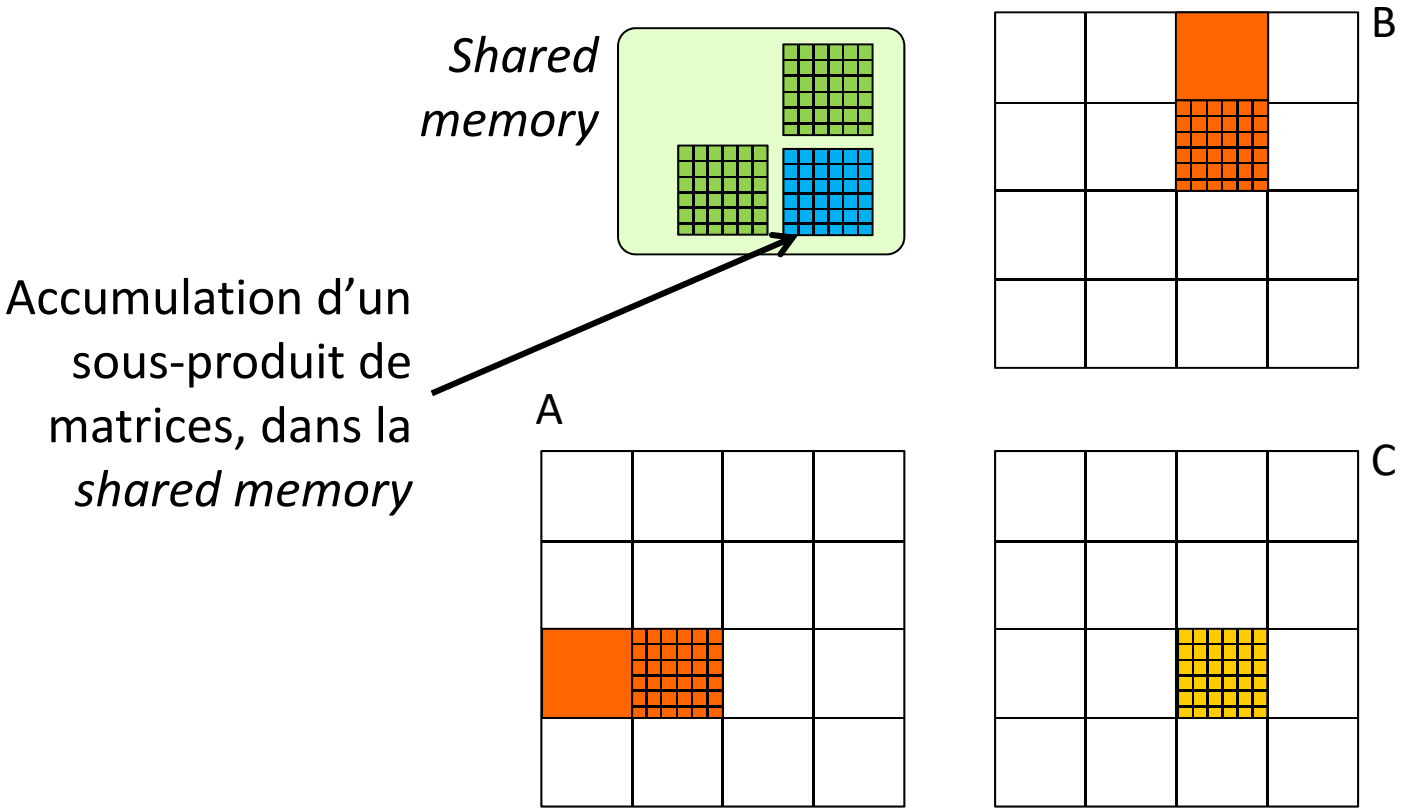


1 – No boundaries problem

Blocs 2D de threads (*kernel k2*)

MatrixSide = *k.BlockSize_xy*

Step 1.b



1 – No boundaries problem

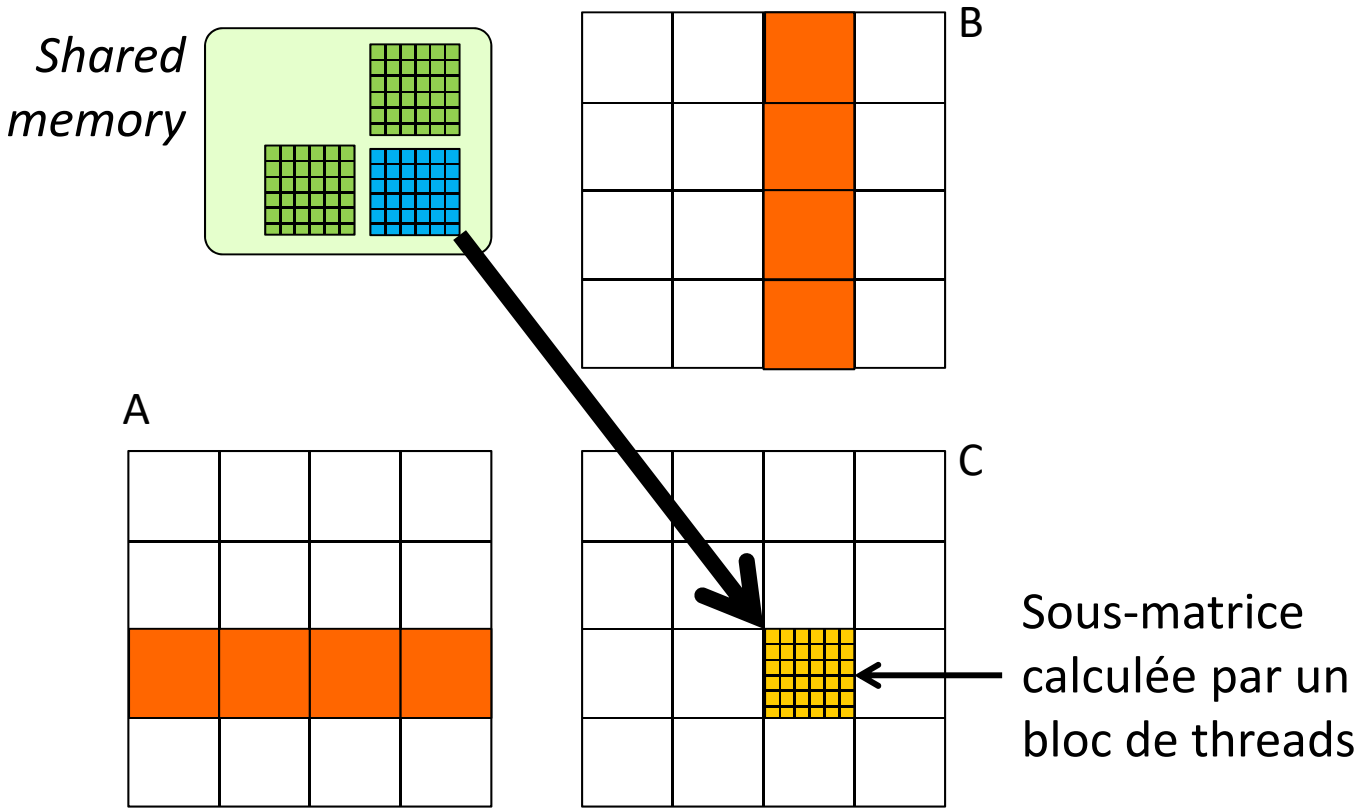
Blocs 2D de threads (*kernel k2*)

MatrixSide = *k.BlockSize_xy*

Last step

Renvoie de la sous-matrice résultat dans la mémoire globale.

Accès coalescent à la mémoire globale.

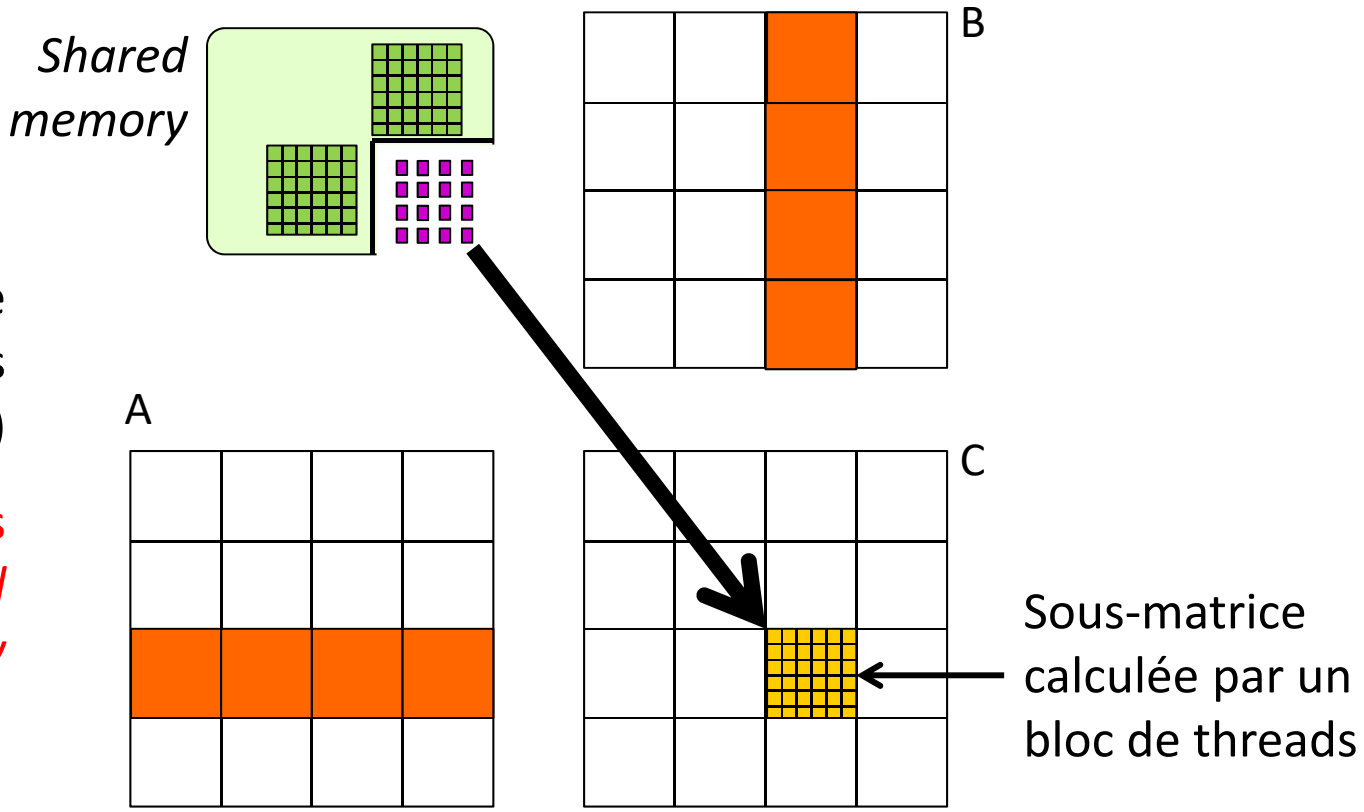


1 – No boundaries problem

Blocs 2D de threads (*kernel k2*)

MatrixSide = k.BlockSize_xy

Avec des registres pour stocker les résultats intermédiaires



Les threads ne partagent pas leurs résultats (C_{ij})

→ Inutile de les stocker en *shared memory*

1 – No boundaries problem

Blocs 2D de threads (*kernel k2*)

MatrixSide = *k.BlockSize_xy*

Implantation (presque) simple si :

- Matrices carrées de : **`SIZE × SIZE`**
- Blocs carrés de : **`BSXY × BSXY`**
- **`SIZE = q × BSXY`** (pavage exact du problème par les blocs)

Algorithme :

- Reset d'une sous-matrice de résultats en *shared memory*
- **`SIZE/BSXY`** steps :
  - Step x.a : monter sous-matrices A et B en *shared memory*
  - Barrière de synchronisation entre threads du bloc
  - Step x.b : faire un sous-produit de matrices en *shared memory*
  - Barrière de synchronisation entre threads du bloc
- Retour de la sous-matrice de résultat en mémoire globale

TD-TP-3: programming with *shared memory*

1 – No boundaries problem

Blocs 2D de threads (*kernel k2*)

MatrixSide **=** *k.BlockSize_xy*

Q 1.1 Implantez le kernel k2 :

- Utilisez la *shared memory* pour stocker les blocs de A et de B,
- Pour les éléments de C vous pouvez utiliser aussi la *shared memory* ou bien continuer à utiliser des registres
- Définissez BLOCK_SIZE_XY_K2 dans main.h
- Considérez que : SIZE = k.BLOCK_SIZE_XY_K2

- Vos accès en mémoire globale sont-ils bien coalescents ?

- Testez et validez votre implantation :
  - en utilisant des blocs de 32x32
  - en calculant une matrice C de 4096x4096 float
  - et en comparant les résultats à ceux du kernel k1.

1 – No boundaries problem

Blocs 2D de threads (*kernel k2*)

MatrixSide = k.BlockSize_xy

Q 1.2 Mesurez les performances du kernel k2 :

- Mesurez les performances sur une matrice de 4096x4096 float
- Comparez au kernel 2D K1 (qui n'utilise pas la *shared memory*)
- Complétez le fichier Excel. Quel est le gain obtenu ?

Q 1.3 Calculez la réduction du nombre d'accès en mémoire globale

- Combien d'accès en mémoire globale fait le kernel k1, en lecture et en écriture (indépendamment du travail des caches L2-L1) ?
  (faites le calcul pour une matrice de SIZExSIZE, puis faites l'application numérique pour SIZE = 4096)
- Combien en fait le kernel k2 ?
  (faites le calcul pour des blocs de taille BSXYxBSXY, puis faites l'application numérique pour BSXY = 32)
- Quel est le facteur de réduction des accès en mémoire globale ?

2 – With boundaries problem

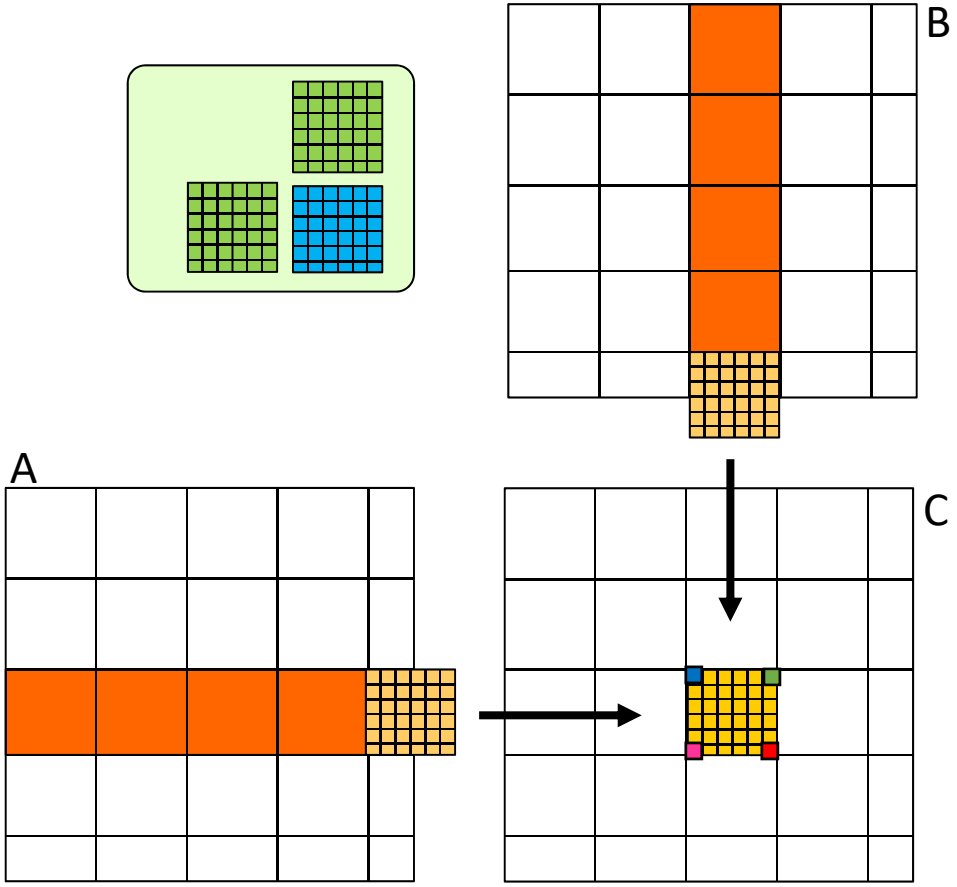
Blocs 2D de threads (*kernel k3*)

MatrixSide ≠ *k.BlockSize_xy*

1 - Quand faut-il « faire le cache » de A ?

2 - Quand faut-il « faire le cache » de B ?

3 - Quand faut-il faire les calculs ?



2 – With boundaries problem

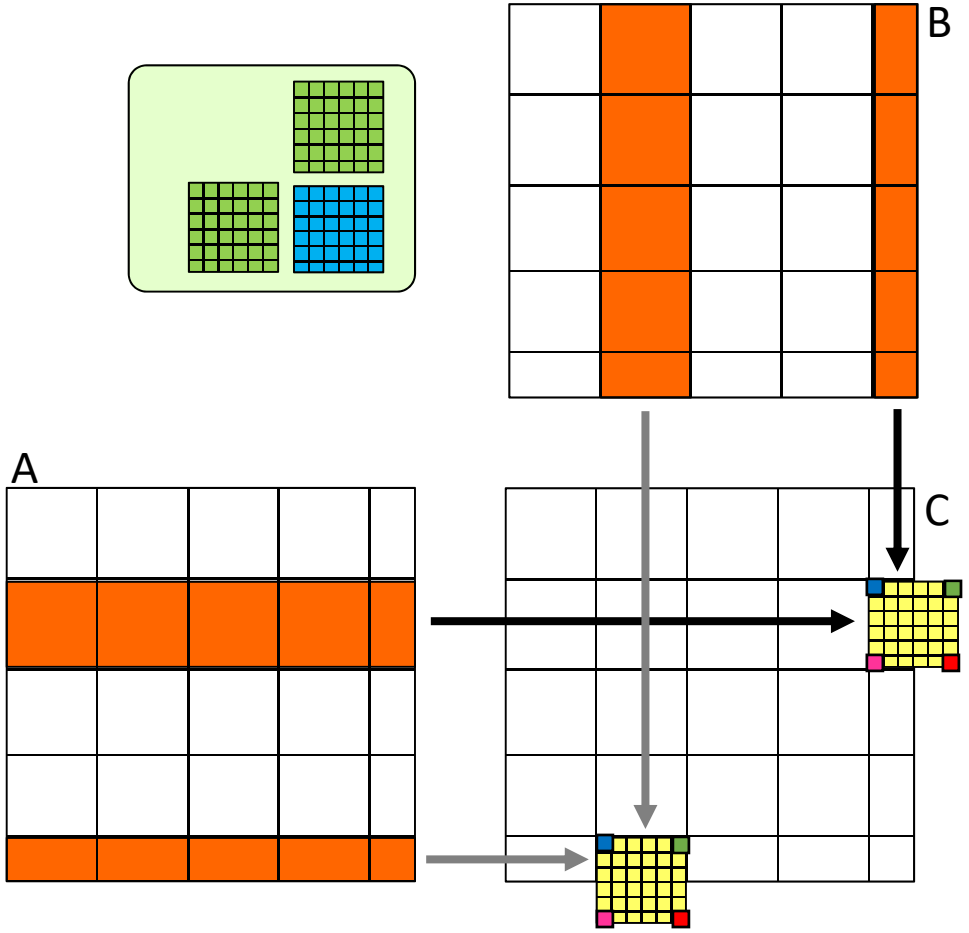
Blocs 2D de threads (*kernel k3*)

MatrixSide ≠ k.BlockSize_xy

1 - Quand faut-il « faire le cache » de A ?

2 - Quand faut-il « faire le cache » de B ?

3 - Quand faut-il faire les calculs ?



2 – With boundaries problem

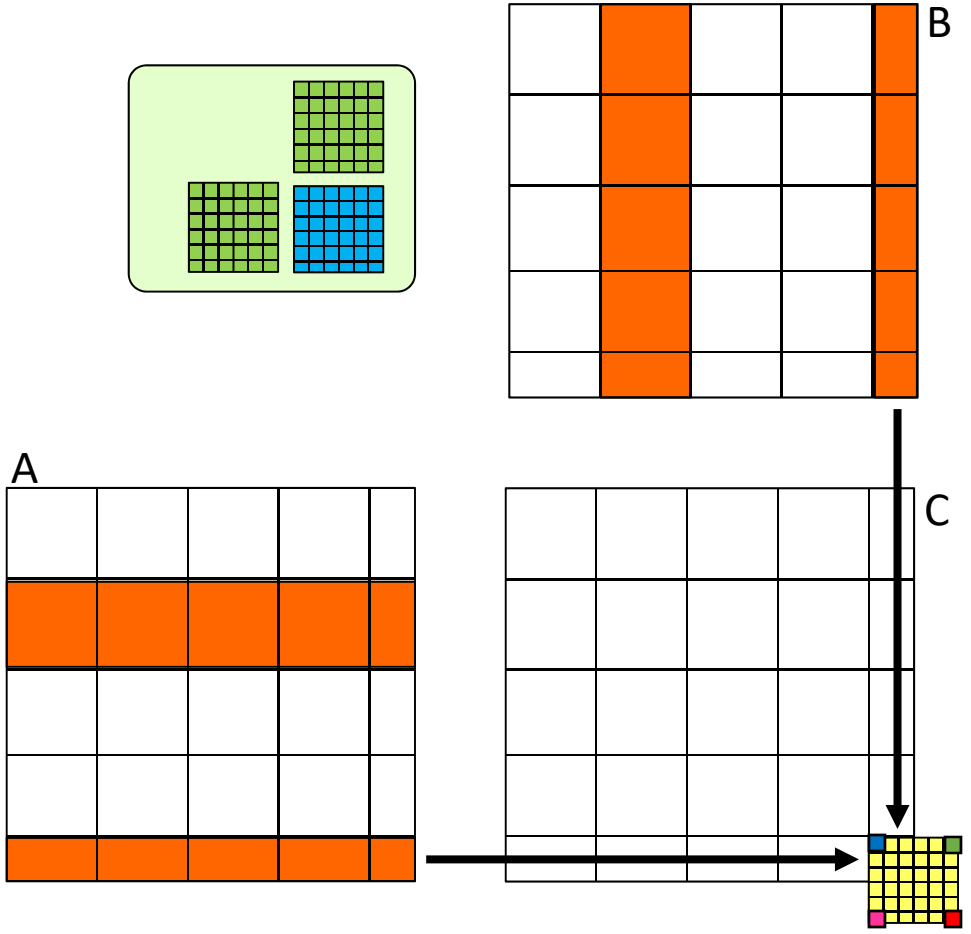
Blocs 2D de threads (*kernel k3*)

MatrixSide ≠ k.BlockSize_xy

1 - Quand faut-il « faire le cache » de A ?

2 - Quand faut-il « faire le cache » de B ?

3 - Quand faut-il faire les calculs ?



2 – With boundaries problem

Blocs 2D de threads (*kernel k3*)

MatrixSide ≠ *k.BlockSize_xy*

Q 2.1 Implantez le kernel k3 :

- Généralisez le kernel k2 en k3, pour le cas ou SIZE n'est PAS un multiple de BLOCK_SIZE_XY_K3 (à définir dans main.h)
- Veillez à prévoir tous les cas représentez sur les slides précédentes.
- Implantez tous les tests nécessaires, et seulement les tests nécessaires! (des tests inutiles pourraient ralentir l'application)

- Testez et validez votre implantation :
  - en utilisant des blocs de 32x32
  - en calculant une matrice C de 4096x4096 float
  - puis une matrice C de 4097x4097 float
  - et en comparant les résultats à ceux du kernel k1.

2 – With boundaries problem

Blocs 2D de threads (*kernel k3*)

MatrixSide ≠ *k.BlockSize_xy*

Q 2.2 Mesurez les performances du kernel k3 :

- Mesurez les performances sur une matrice de 4096x4096 float
- Complétez le fichier Excel

- Comparez au kernel 2D K1 (qui n'utilise pas la *shared memory*)
- Quel est le gain obtenu ?

- Comparez au kernel 2D k2 (qui n'est pas générique)
- Observez-vous un ralentissement mesurable ?

CentraleSupélec

TD-TP-3: programming with *shared memory*

3 – Report & Oral Exam

Documents à produire :

- **Un fichier PDF de 2 pages de résumé** (tout inclus)
  Nommé TP3-CUDA-Nom1-Nom2.pdf
  A envoyer 48h avant votre IO à *Stephane.Vialle@centralesupelec.fr*
- **1+6 slides** de présentation pour votre IO
  Présentés le jour de votre IO
  Prévoir 15 minutes d'exposé et 5' de questions
  Equilibrer les temps de parole dans le binôme

CentraleSupélec

TP-3
CUDA programming with *shared memory*

End