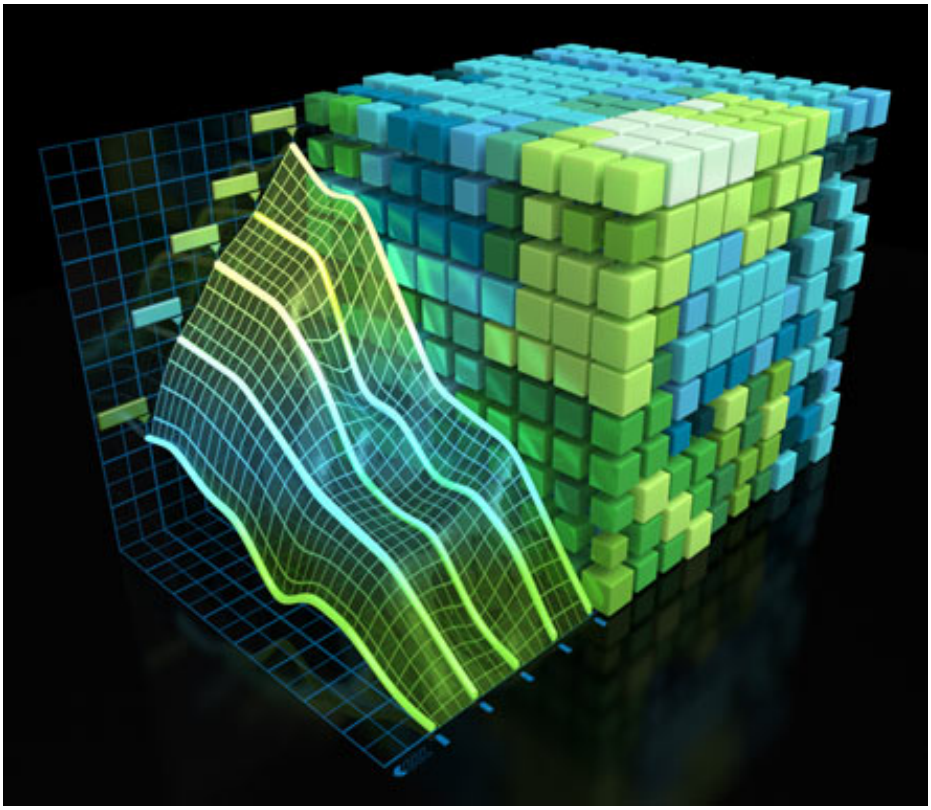
haute école
neuchâtel berne jura
ingénierie
saint-imier le locle porrentruy

Hes·so
Haute Ecole Spécialisée
de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences
Western Switzerland

Cuda

Calcul Tid



Joaquim Stähli

Professeur
Cédric Bilat

Table des matières

1 Contexte 3

2 Problème : 4

3 Approche 4

4 Solutions : 5

4.1 Solution : *rawFull-major* 5

4.2 Solution : *block-row-major Best* 5

5 Formule 7

5.1 Formule : *rowFull-major* 7

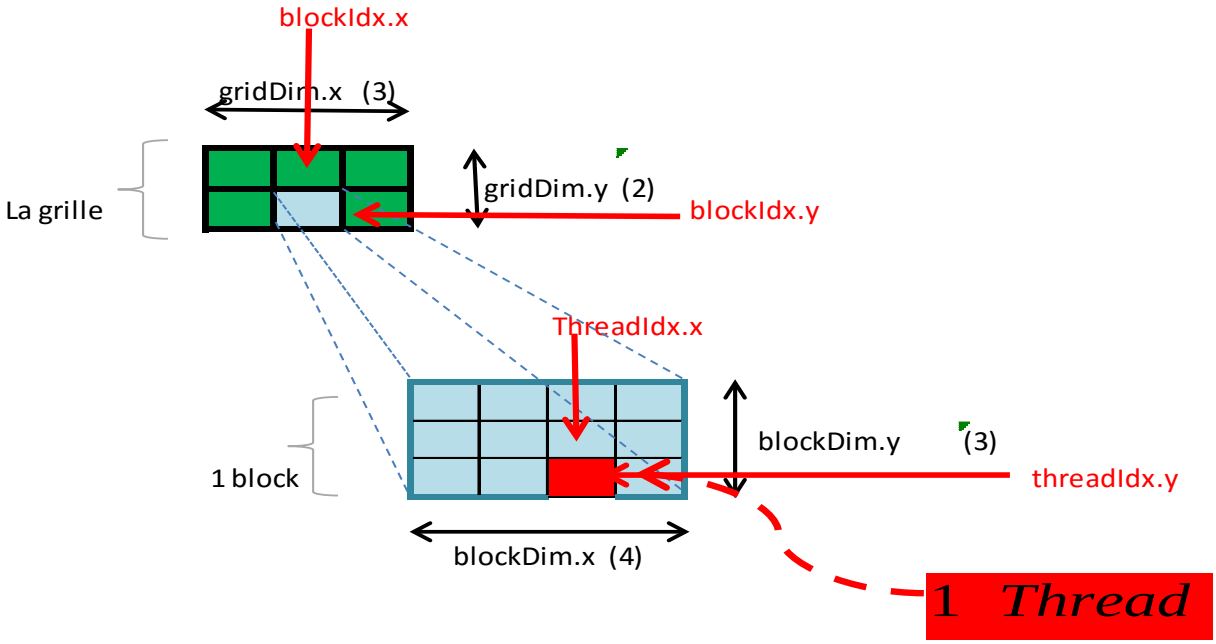
5.2 Formule : *block-row-major Best* 8

1 Contexte

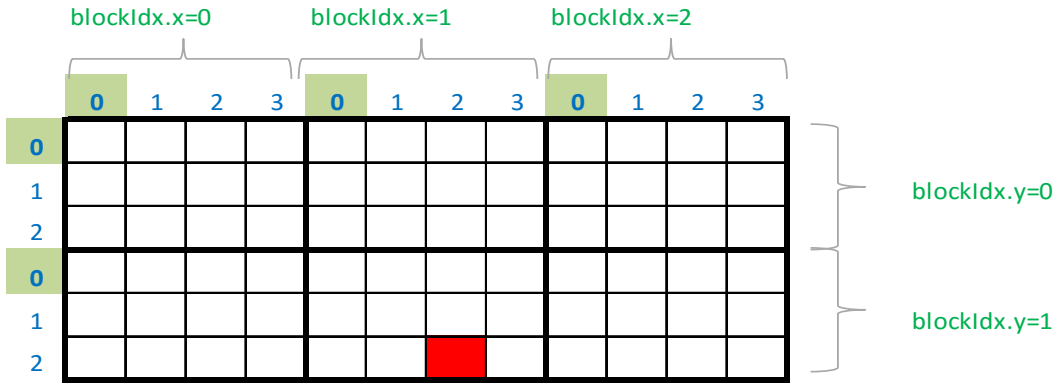
Dans le cas de grille et block 2D, Il y a 4 indices cuda coté serveur représentant un thread

- blockIdx.x
- blockIdx.y
- threadIdx.x
- threadIdx.y

On peut se représenter ces 4 indices comme suit :



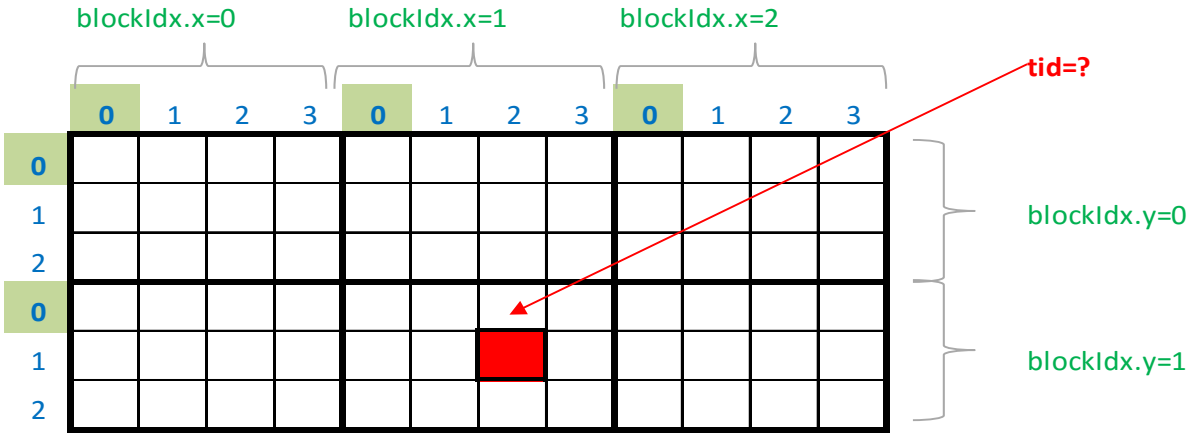
ou en regroupant



Ici chaque case représente un thread. Un thread est donc caractérisé par 4 indices dans cette structure hiérarchique !!

2 Problème :

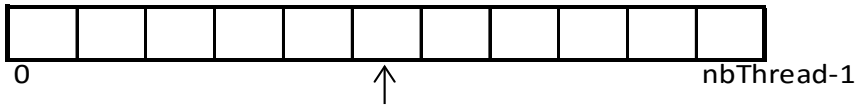
Un thread est représenté par 4 indices. On aimerait pouvoir identifier un thread de manière unique avec 1 indice seulement !



3 Approche

Il suffit de trouver un algorithme de transformation, qui associe à la grille 2D, un tableau 1D.

Le tableau 1D étant linéaire, le tid (Thread Index) utilisé sera celui induit par la position du thread dans le tabelau 1D.



$$tid = tid\left(blockIdx.y, blockIdx.x, threadIdx.y, threadIdx.x\right)$$

Attention

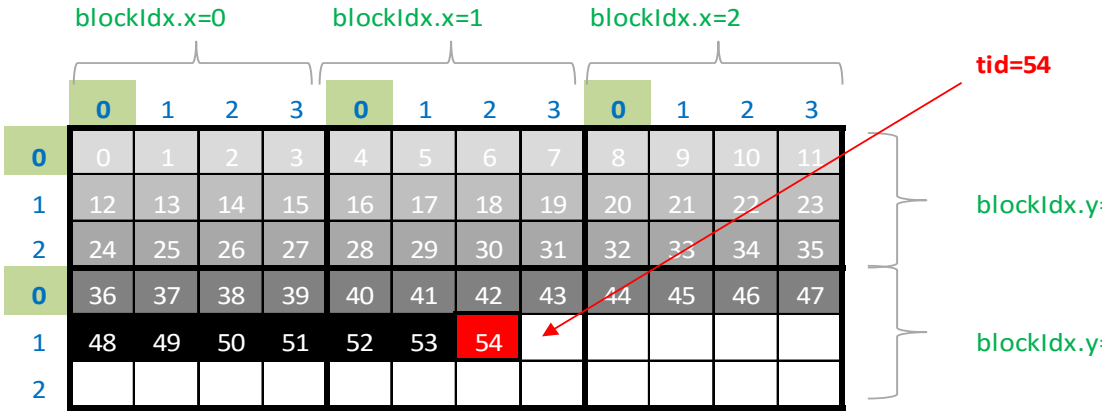
(A1) Il n'y a pas qu'une manière de transformer la grille en tableau1D.

(A2) Selon la transformation que l'on choisit, les performances du code Cuda peuvent varier énormément (en cause les transactions mémoires)

4 Solutions :

4.1 Solution : *rawFull-major*

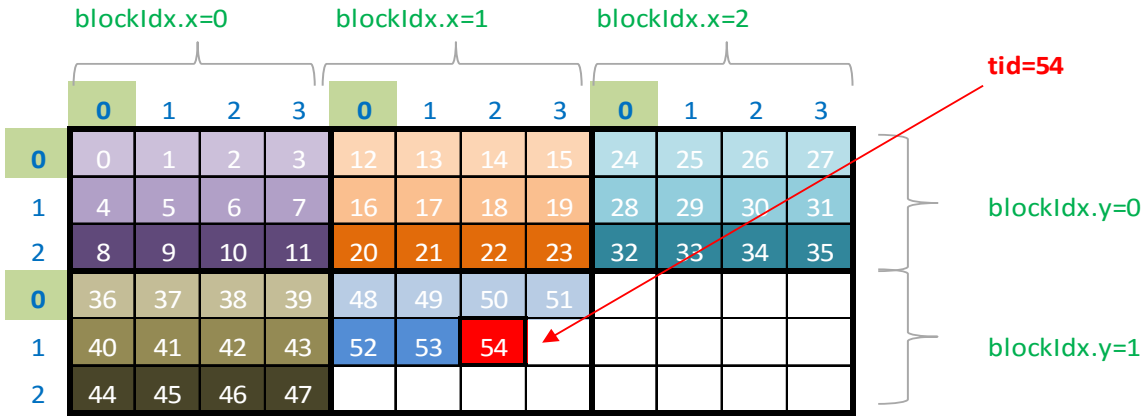
On vectorise la grille de manière ***rawFull-major***, comme l'illustre le schéma suivant :



La colorisation en niveau de gris induit la vectorisation.

4.2 Solution : *block-row-major* *Best*

On vectorise la grille de manière ***block-row-major***, comme l'illustre le schéma suivant :



La colorisation en niveau de gris induit la vectorisation.

On comprend qu'il y a beaucoup d'autres manières d'associer un tid, par exemple :

- *Block-colon-major*
- ….

Les plus optimum en termes de transaction de transaction mémoire sont celle de type Block-XXX-major. Comme C et Cuda sont *raw-major*, on choisit dans ce cours d'utiliser :

block-raw-major

5 Formule

5.1 Formule : *rowFull-major*

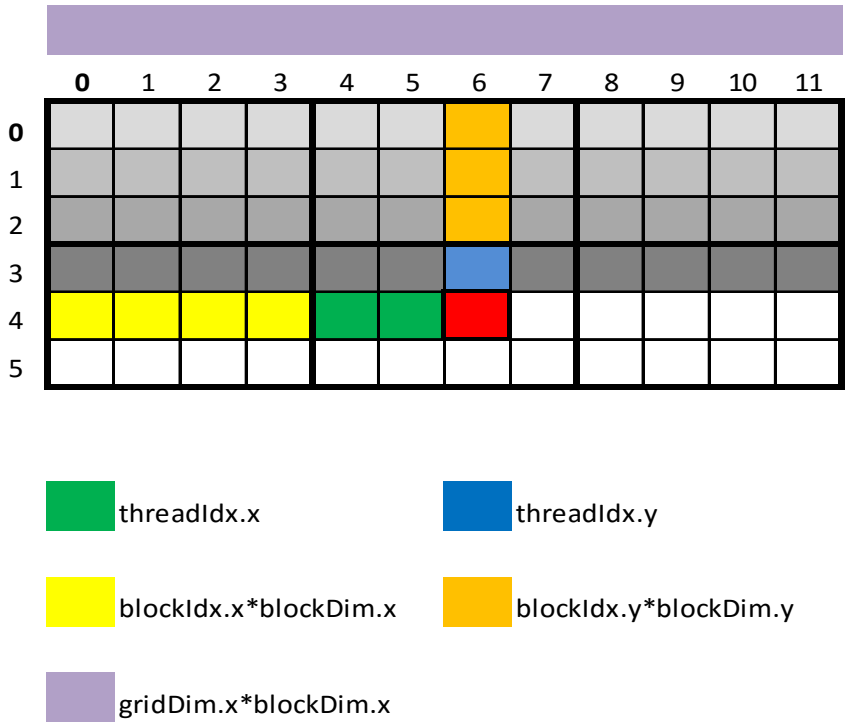
Input *(server side)*

Dans le cas 2D, les 4 variables à disposition sont :

- *blockIdx.x*
- *blockIdx.y*
- *threadIdx.x*
- *threadIdx.y*

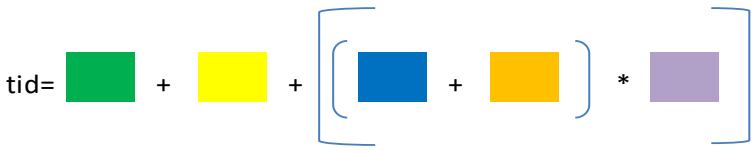
Echauffement

Commençons par calculer les quantités induites par les couleurs ci-dessous



Observons que les formules sont les mêmes sur l'axe des x et des y. Seul l'attribut x ou y change !

Solution



5.2 Formule : *block-row-major Best*

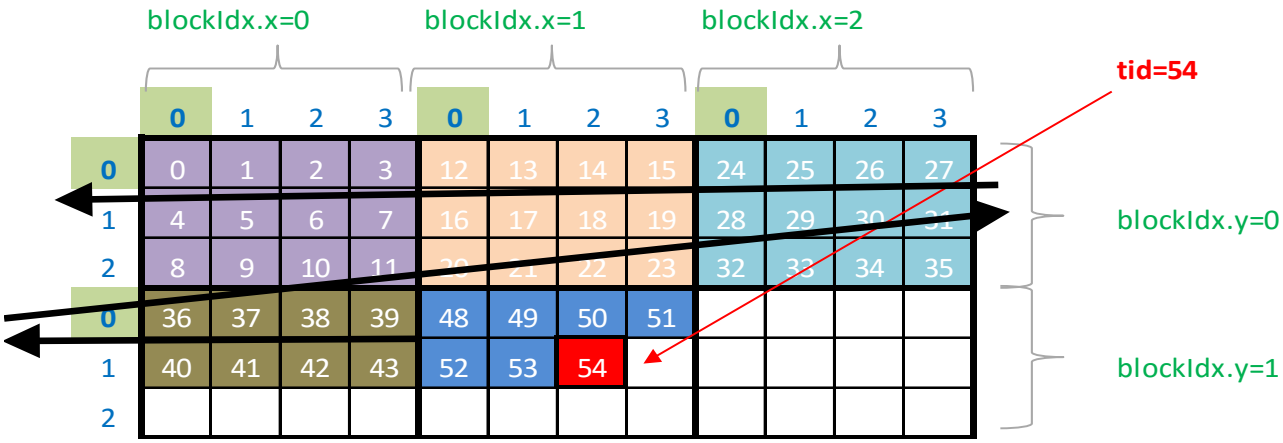
Input *(server side)*

Dans le cas 2D, les 4 variables à disposition sont :

- *blockIdx.x*
- *blockIdx.y*
- *threadIdx.x*
- *threadIdx.y*

Rappel

La vectorisation (ou indexation) est induit par le parcours des blocks selon la féche noire ci-dessous. A l'intérieur des blocks, le parcours est raw-major.

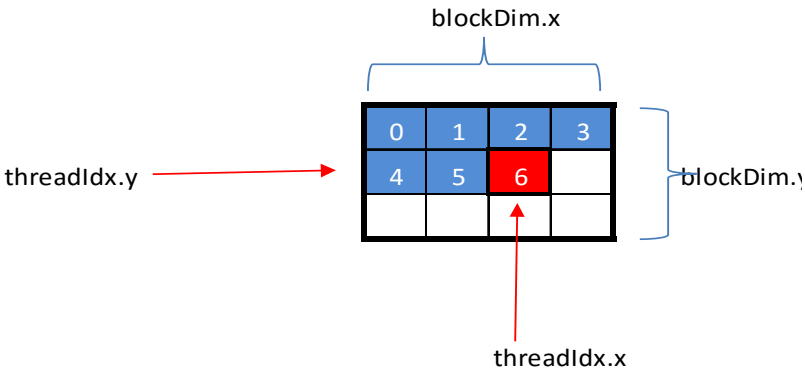


Il s'agit en fait d'un parcours

- intra-block raw major
- inter-block raw-major

Solution

On peut commencer par calculer le *tid local au block* courant (bleu) :



Il est trivial que

$$tid_{local} = threadIdx.x + (threadIdx.y * blockDim.x)$$

Ensuite on calcule le nombre de block à notre gauche selon le parcours fléché ci-dessus :

$$\#blockAvant\ (meme\ rangée) = blockIdx.x$$
$$\#blockAvant\ (au\ dessus) = blockIdx.y * gridDim.x$$

avec la notation

signifie “nombre de “

Comme par block on a

$$\#threadBlock = blocDim.x * blocDim.y$$

Au final

$$tid = tidLocal + [\#blockAvant * \#threadBlock]$$

Conclusion

En regroupant :

$$\boxed{\begin{aligned} tid = {} & threadId.x + (threadIdx.y * blockDim.x) \\ & + [(blockIdx.x + blockIdx.y * gridDim.x) * (blocDim.x * blocDim.y)] \end{aligned}}$$