Modèles et techniques en programmation parallèle hybride et multi-cœurs

Utilisation des accélérateurs de calcul

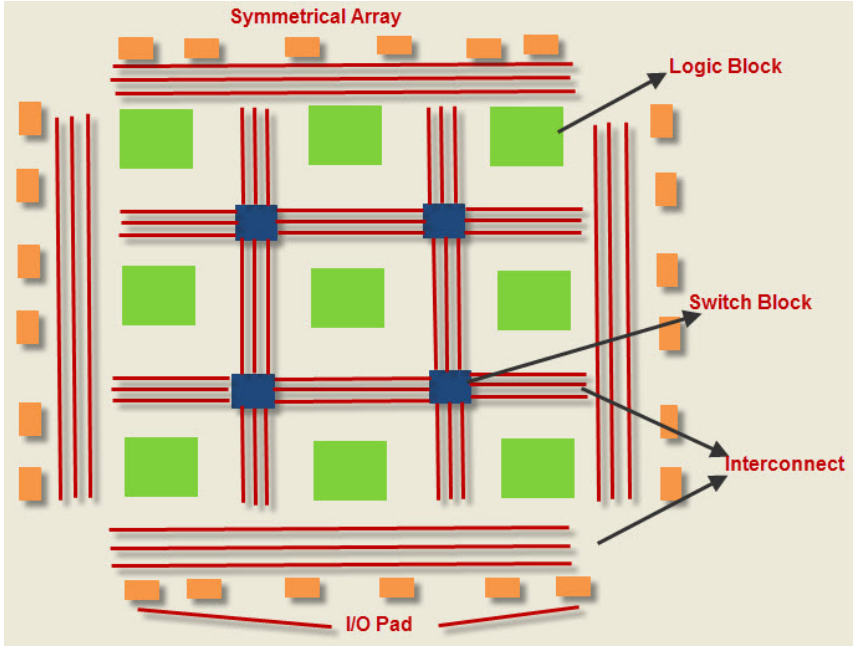
Marc Tajchman

CEA - DEN/DM2S/STMF/LMES

14/01/2022

Circuits FPGA

Les circuits **FPGA** (**F**ield-**P**rogrammable **G**ate **A**rray) est constitué d'un ensemble de circuits électroniques (Logic Block dans la figure) connectés entre eux et dont les paramètres et les connexions (Switch Block, Interconnect) peuvent être changées par l'utilisateur.



Exemple de matrice de circuits FPGA

La façon d'utiliser les FPGA dépend de chaque dispositif.

En général, l'utilisateur écrit la description du système avec un langage HDL (Hardware Description Language).

Exemple:

```
library ieee;
use ieee.std_logic_1164.all;

entity E is
port (
    I1:in std_logic;
    I2:in std_logic;
    O:out std_logic
);
end E;
architecture rtl of E is
signal and_gate: std_logic;
begin
 and_gate <= I1 and I2;
 O <= and_gate;
end rtl;
```

On compile ensuite le code en langage HDL qui configure les circuits électroniques et on obtient un système qui peut exécuter seulement l'algorithme contenu dans le programme HDL.

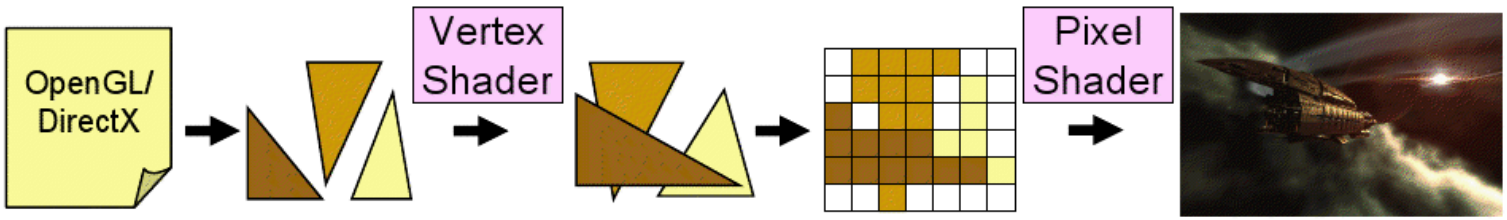
L'exécution est très rapide (pas de décodage des instructions) mais pour exécuter autre chose, il faut recompiler un autre code HDL.

Utilisé principalement pour des dispositifs très spécialisés (applications embarquées dans les domaines aéronautique, médical, audio, gps, etc)

Catégorie voisine : **ASIC** (**A**pplication-**S**pecific **I**ntegrated **C**ircuit), encore plus rapides, mais non programmables, les circuits qui exécutent l'algorithme choisi, sont fixés une fois pour toute.

Utilisation des GPU

Les GPU (**G**raphics **P**rocessing **U**nits ou cartes graphique) ont été conçus pour faire le plus rapidement possible les calculs nécessaires à l'affichage : affichage pixels couleurs, tracé de formes, projection 3D sur l'écran, etc.



Pour programmer les traitements graphiques par les GPU, plusieurs interfaces spécialisées existent (OpenGL, DirectX, Metal, ...).

Ces traitements sont souvent très parallélisables et donc les constructeurs de cartes graphiques y incluent beaucoup d'unités de calcul (ou cœurs) (plusieurs milliers actuellement).

Jusque 2000, la seule façon de programmer les GPU est l'utilisation de OpenGL, DirectX, etc.
Utilisation principalement pour des traitements graphiques.

La puissance de calcul des GPU a conduit à essayer de les utiliser pour des traitements non graphiques. On parle alors de GPGPU (General Purpose computing on Graphics Processing Units).

A partir de 2000, apparaissent les modèles de programmation adaptés aux (GP)GPU : Ati Stream (ATI-AMD), Cuda (Nvidia, "Compute Unified Device Architecture"), puis OpenCL (interface générique, disponible chez Apple, Intel, AMD, Nvidia).

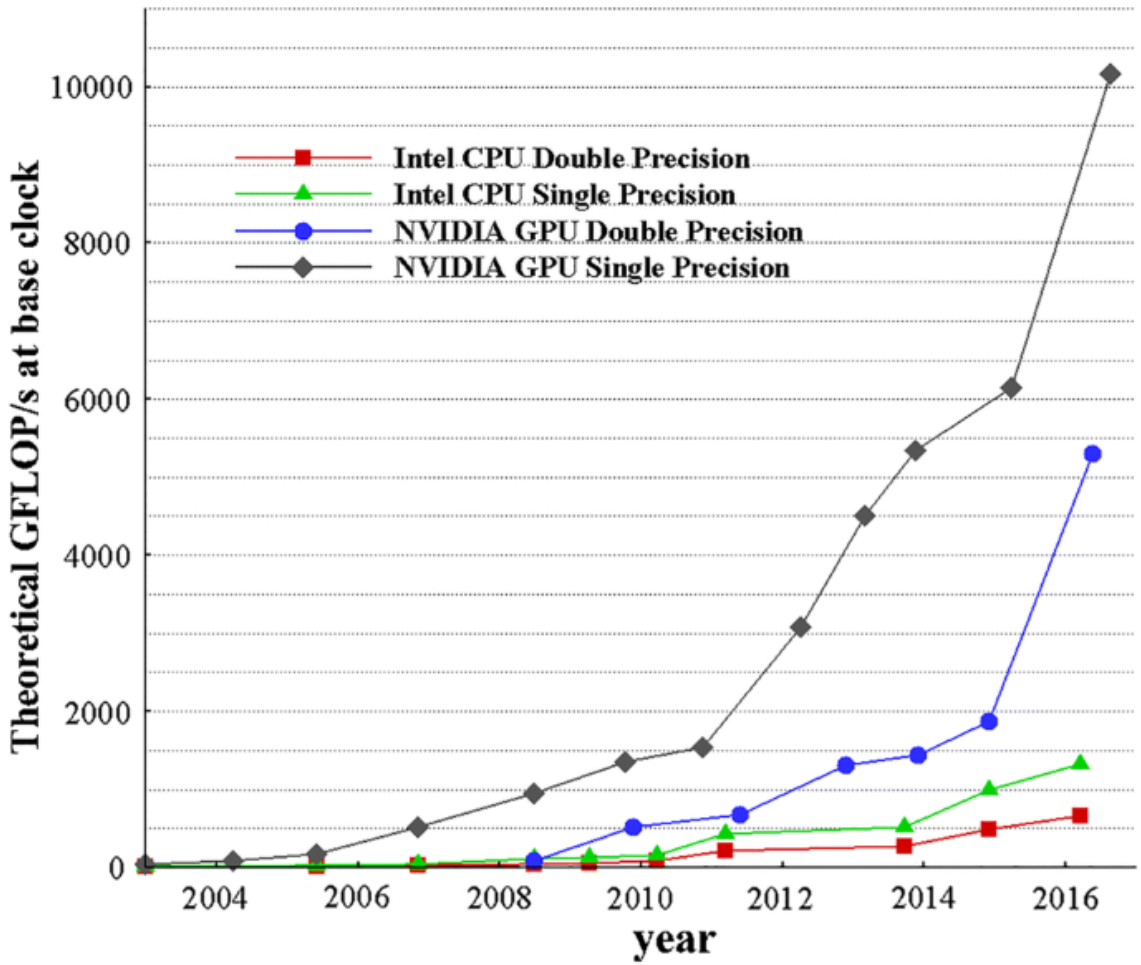
Les 2 modèles les plus courants actuellement sont Cuda et OpenCL

Principales différence entre les CPU et les GPU (situation actuelle) :

	CPU	GPU
nombre de cœurs	😐 $\sim$ 4 à 64	🙂 $\sim$ 1000 à 10000
chaque cœur	🙂 $+$ performant	☹ $-$ performant
jeu d'instructions	🙂 plus général	😐 spécialisé calcul
bande passante	☹ $-$ importante	🙂 $+$ importante
latence	🙂 $+$ petite	☹ $+$ grande

Ici, latence, bande passante : durée initialisation et quantité échangée par seconde entre la mémoire et le processeur.

La puissance de calcul théorique (nombre de cœurs $\times$ performance d'un cœur) des GPU dépassent celle des CPU : les cœurs GPU sont moins puissants que les cœurs CPU mais (beaucoup) plus nombreux.



Pourquoi une telle différence ?

- Les CPU exécutent tous les traitements, parallèles ou non

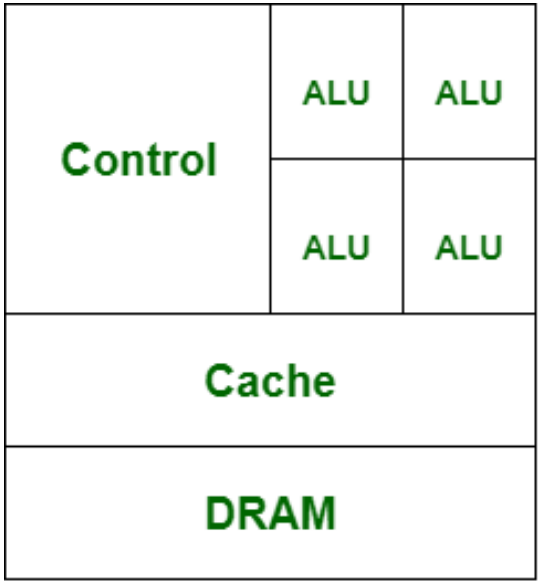
  Par conception, les CPU sont optimisés pour des calculs séquentiels : beaucoup de mémoire cache, plusieurs additionneurs/multiplicateurs, prédiction de branches, exécution d'instructions "dans le désordre" (out-of-order)

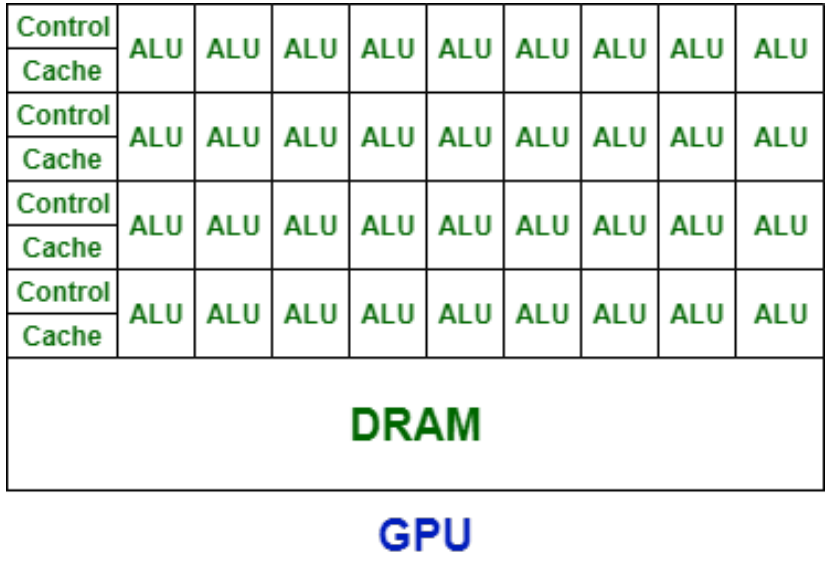
- Les GPU supposent que le degré de parallélisme est élevé

  Par conception, les GPU maximisent la bande passante utilisable par beaucoup de threads, moins de mémoire cache (latence compensée par le multithreads massif), circuits de contrôle partagés entre les threads

Structure interne comparée (schématique):

- ALU : cœur
- Control : contrôleur mémoire
- Cache : mémoire cache
- DRAM : mémoire de travail

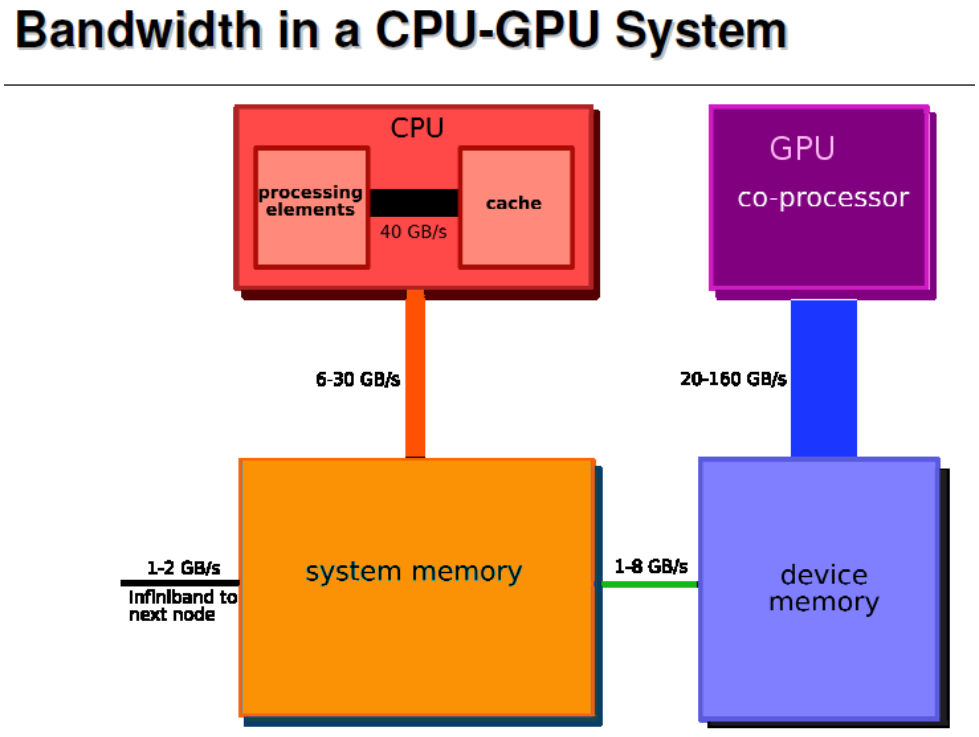




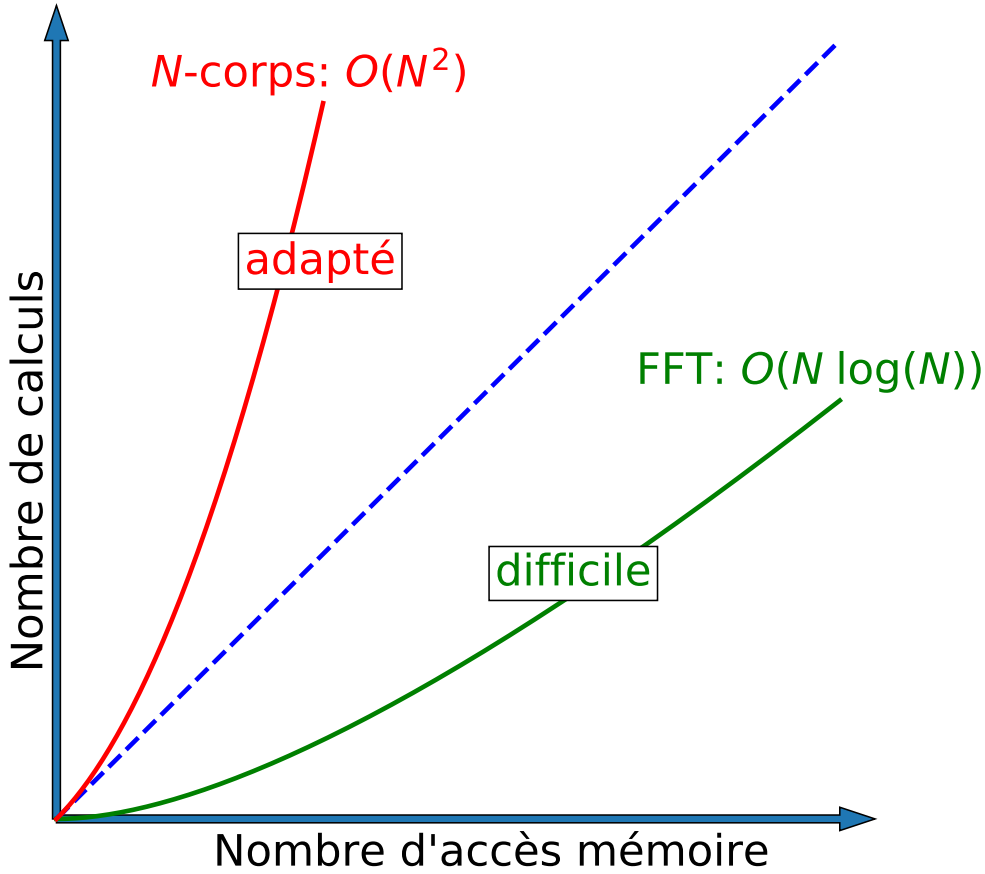
Actuellement, un GPU doit toujours être associé à un CPU (pour gérer le clavier, la souris, le réseau, les disques, etc.).

Chacun a sa propre mémoire de travail et donc il faut pouvoir transférer des données entre tous les éléments.

Comparaison des vitesses de transfert entre les unités de calcul et les différentes mémoires:



Certains types de problèmes sont bien adaptés au calcul sur GPU, d'autres (beaucoup) moins



Problème N-corps: N objets massifs exercent entre eux des forces de gravité (exemple: systèmes d'étoiles)

Sur une seule machine, pour :

calculs séquentiels ou faiblement // :	utiliser le CPU
calculs fortement // :	utiliser le GPU

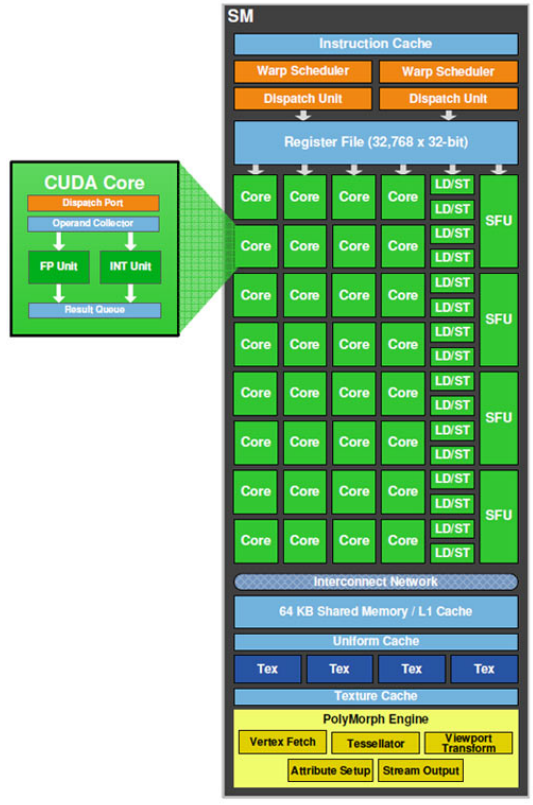
A adapter à chaque cas !!

Les résultats ne seront pas toujours exactement les mêmes sur CPU et sur GPU:

- Jusque $\sim$ 2000, beaucoup de GPU calculaient seulement en simple précision (32 bits) : pas besoin de 15 décimales pour calculer la couleur de pixels.
- Depuis, on a du 64 bits (double précision) sur GPU, **mais** pas toujours les mêmes précisions entre le GPU et le CPU pour les additions/multiplications/divisions/..., les modes d'arrondi ne sont pas toujours identiques

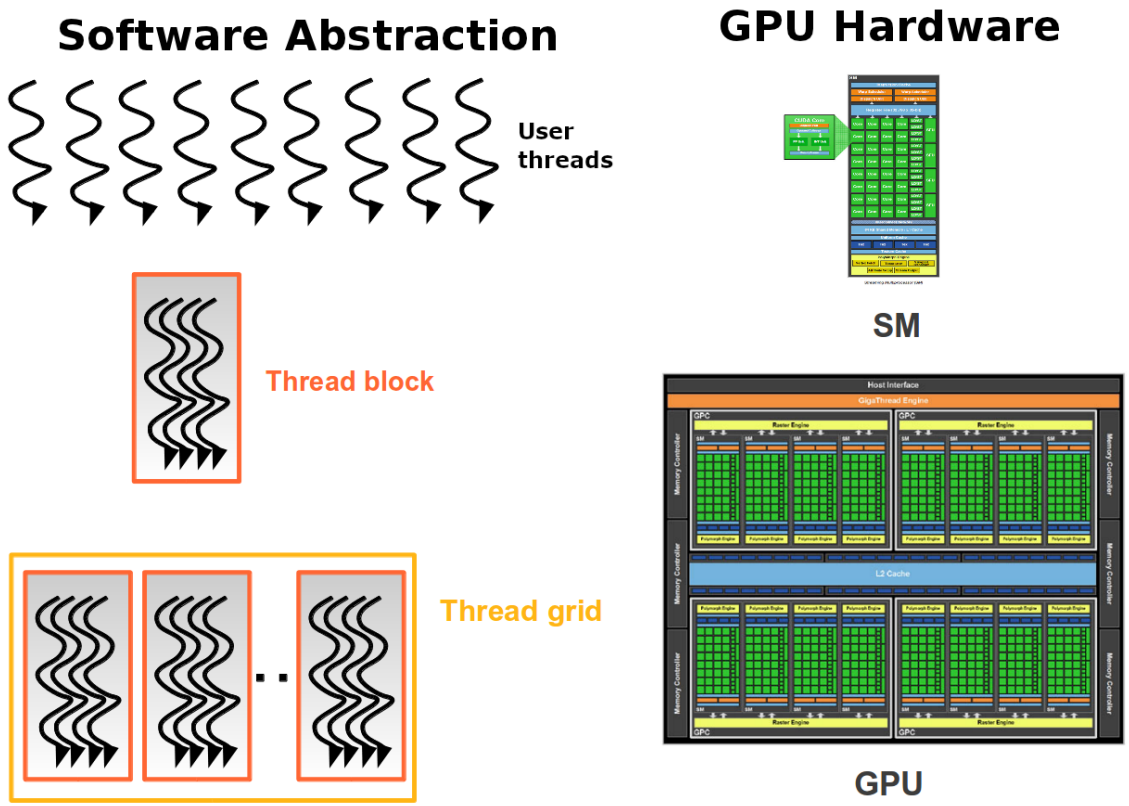
Structure d'un GPU (exemple chez Nvidia, similaire pour les autres constructeurs).

Les cœurs du GPU sont groupés dans des "streaming processors" (32 cœurs)



Streaming Multiprocessor (SM)

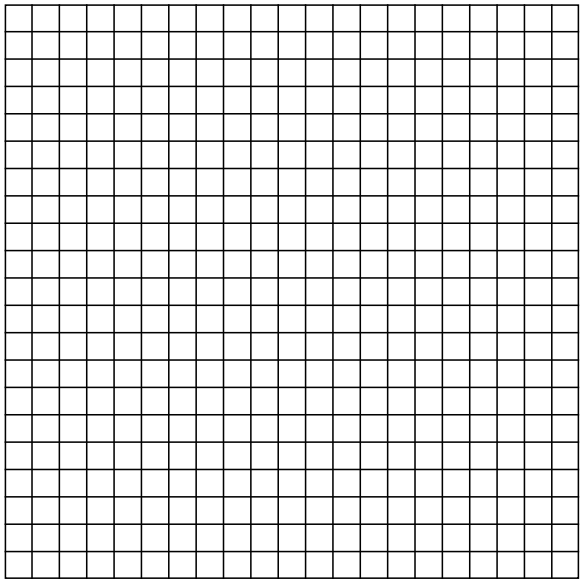
Modèle de programmation CUDA : les threads utilisateurs sont groupés en blocs de threads, organisés en grilles de blocs.

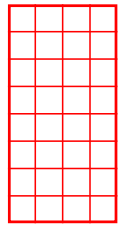


Les blocs de threads sont dans une file d'attente. Le premier bloc de threads de la file est envoyé vers un "streaming processor" disponible. Quand un streaming processor a fini d'exécuter un bloc de threads, il est disponible pour le bloc suivant de la file.

Exemple: addition de matrices $N \times N$
Choix d'une taille de bloc 4×8 et recouvrement des matrices par une grille de blocs:

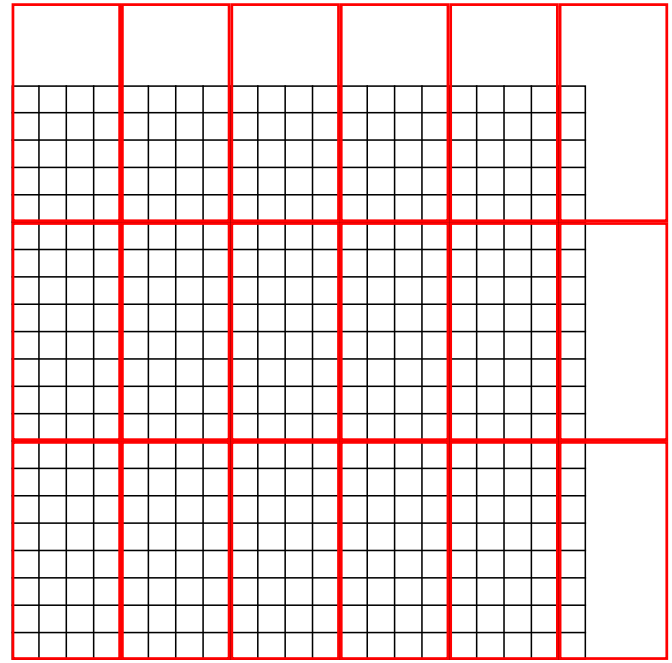
Matrice





Bloc de 8 x 4 threads

Répartition du calcul des coefficents dans les blocs de threads



Il faut essayer de “remplir le plus possible la carte graphique” pour utiliser toute sa puissance :

- chaque streaming processor contient 32 cœurs, donc il faut que chaque bloc de threads contienne un multiple de 32 threads (en tout cas une puissance de 2)
- un GPU contient plusieurs streaming processors, donc il faut assez de blocs pour faire travailler tous les streaming processors
- il faut choisir une taille de blocs qui minimise les blocs incomplètement remplis (voir figure page précédente)

Par exemple : pas de blocs 2 x 2 ou 4 x 4 (trop petits), pas de blocs 1024 x 1024 (trop grands : il y aura probablement pas assez de blocs pour faire travailler tous les streaming processors).
On prend souvent 8 x 16, 16 x 16 ou 32 x 16

Modèle de programmation OpenCL similaire à celui de Cuda.
Les concepts sont similaires, exemples :

- blocks (Cuda) - work group (OpenCL)
- thread (Cuda) - work item (OpenCL)

Commentaires sur un exemple écrit à l'aide de Cuda ou d'OpenCL