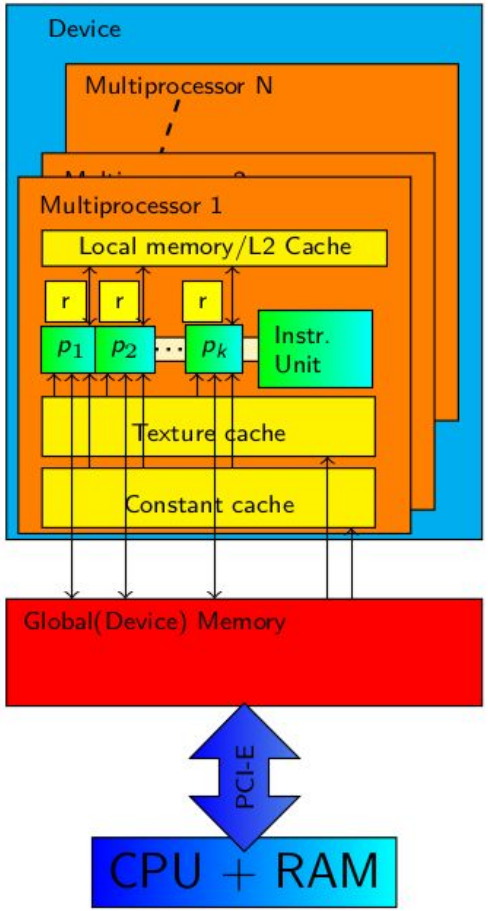
RÉPUBLIQUE
FRANÇAISE
Liberté
Égalité
Fraternité

ONERA
THE FRENCH AEROSPACE LAB

GPGPU

Détail de l'architecture GPGPU

- GPGPU : Ensemble de N petites unités SIMD indépendantes partageant une mémoire global commune : N multiprocesseurs;
- Multiprocesseur : Petite unité SIMD avec :
  - k ALU synchronisés;
  - 1 décodeur d'instruction;
  - Trois mémoires partagées pour tous les ALUs (dont deux mémoires caches)
  - R registres distribués parmi les ALUs (locales à chaque *thread*) (Exemple Maxwell : 65536)



C étendu

- **Nouv. déclarations** : global, device, shared, local, constant

```
__device__ float filter[N];
__global__ void  convolve(float* image) {
__shared__ float region[M];
```

- **nouveaux mots clefs** : threadIdx, blockIdx

```
region[threadIdx] = image[i];
```

- **Intrinsics** : __syncthreads

```
__syncthreads(); image[j] = result;
```

- **API d'exécution** : Memory, symbol, execution management

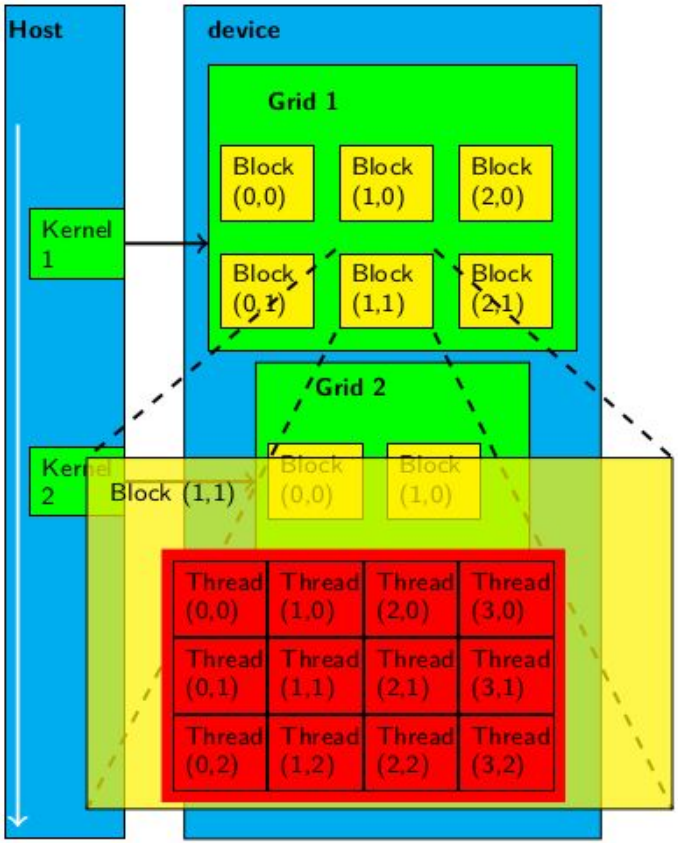
```
void* myImg = cudaMalloc(bytes);// Alloue memoire sur GPU
```

- **Exécution de fonction**

```
convolve<<<100,10>>>(myImg);// 100 blocs de 10 threads
```

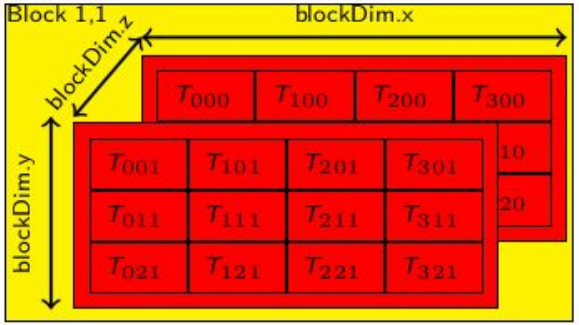
Distribution des threads : grilles et blocs

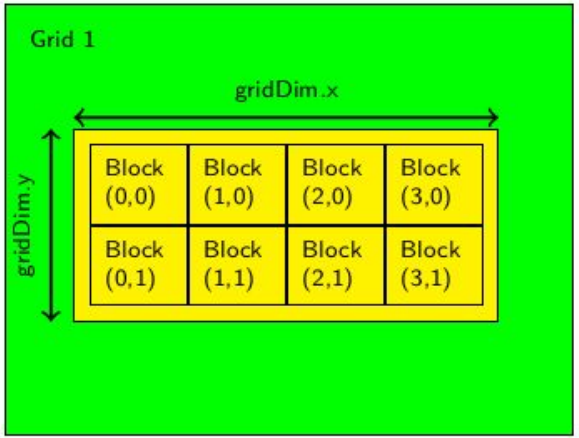
- Un noyau est exécuté comme une grille de blocs de thread
  - Tous les threads partagent le même espace de mémoire de donné
- Un bloc de threads est un ensemble de threads qui peuvent coopérer les uns les autres en :
  - synchronisant leur exécution
  - partageant leurs données à travers une mémoire partagée rapide
- Deux threads provenant de deux blocs différent ne peuvent pas coopérer :
  - Opérations atomiques



Mots clefs pour les blocs et les threads

- Mots clefs pour les blocs :
  - threadId.[x,y,z] définit la position du thread dans le bloc;
  - blockDim.[x,y,z] définit les dimensions du bloc.
- Mots clefs pour les grilles :
  - blockId.[x,y,z] définit la position du bloc dans la grille
  - gridDim.[x,y,z] définit les dimensions de la grille





Thread ID

L'ID d'un thread dans un bloc est :

```
t_id = threadIdx.x +
       threadIdx.y*(blockDim.x) +
       threadIdx.z*(blockDim.x*blockDim.y);
```

- `threadIdx.[x,y,z]` : Indice du thread dans la dimension x,y,z
- `blockDim.[x,y,z]` : Taille du bloc dans la dimension x,y,z

TD4

- Principe
  - tous les threads exécutent le même code (principe du SIMD)
  - on les différencie simplement avec leur thread_id
  - Attention à bien identifier ce qui est en mémoire GPU et CPU

- En pratique :
  - Penser et programmer le « kernel » pour un élément (ou pixel)
  - Allouer et copier les input data sur le GPU (cudaMemcpy HostToDevice)
  - Lancer sur une grille :
    - `ma_fonction <<<DimGrid, DimBlock>>>(…) ;`
  - Récupérer le résultat (cudaMemcpy DeviceToHost)

pycuda

- Pycuda facilite une bonne partie de l'implémentation
- … et rend plus obscure d'autres parties