

# Programmation CUDA

## Mémoire partagée, divergence, synchronisation

Oguz Kaya

Maître de Conférences  
Université Paris-Saclay et l'Équipe ParSys du LRI, Orsay, France

# Objectifs

- Zoom sur l'architecture mémoire d'un GPU
- Utilisation de la “shared-memory” pour accès mémoire rapide
- Synchronisation de threads
- Concept de divergence pour la gestion efficace des branchements

# Outline

- 1 Architecture mémoire d'un GPU
- 2 Utilisation de la mémoire partagée
- 3 Divergence

- 1 Architecture mémoire d'un GPU
- 2 Utilisation de la mémoire partagée
- 3 Divergence

# Architecture mémoire d'un GPU

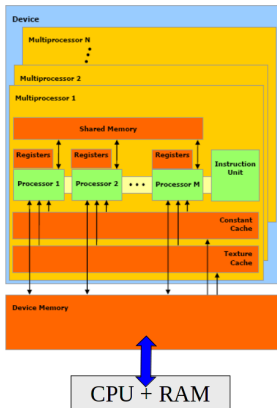
Il y a beaucoup de types de mémoire dans un GPU.



- Mémoire globale (RAM): 8-48Go, 500-2000 Go/s, 300-400 cycles par accès
- Registres: 65536/SM, accès immédiat (1/cycle)
- Cache L1: 64-192Ko/SM,  $\approx 4$  cycles par accès
- Cache L2: 8-40Mo, latence/bande passante pire que L1
- Constant cache: Mémoire constante par SM

# Architecture mémoire d'un GPU

Il y a beaucoup de types de mémoire dans un GPU.



- Mémoire partagée (shared memory):
  - Partagée par tous les threads d'un bloc
  - 32-128Ko/SM
  - Technologie pareil que cache L1
  - Latence  $\approx 4$  cycles/accès
  - Bande passante comparable aux registres
  - Pas besoin de coalescence (aussi performant)
  - Peut se concurrencer avec la capacité du L1 (e.g., unified L1+shared memory)

- 1 Architecture mémoire d'un GPU
- 2 Utilisation de la mémoire partagée**
- 3 Divergence

# Exemple: Calcul de puissance dans un tableau

Étant donné un tableau  $A[N]$  et un entier  $k$ , mettre  $A[i]$  au  $A[i]^k$ .

```
#include <stdio>
#include "cuda.h"

#define N 1024
#define BLOCKSIZE 128
float A[N];

__device__ float dA[N];

__global__ void powerArray(int n, int k)
{
    int i = threadIdx.x + blockIdx.x * blockDim.x;
    if (i < n) {
        float c = 1.0;
        for (int j = 0; j < k; j++) { c *= dA[i]; }
        dA[i] = c;
    }
}

int main(int argc, char **argv)
{
    // Initialisation
    for (int i = 0; i < N; i++) { A[i] = i; }
    // Copier le tableau vers le GPU
    cudaMemcpyToSymbol(dA, A, N * sizeof(float), 0,
        cudaMemcpyHostToDevice);
    int blockSize = 128;
    int numBlocks = N / blockSize;
    if (N % blockSize) numBlocks++;
    powerArray<<<numBlocks, blockSize>>>(N, 4);
    // Recopier le tableau vers le CPU
    cudaMemcpyFromSymbol(A, dA, N * sizeof(float), 0,
        cudaMemcpyDeviceToHost);
    printf("%1f\n", A[2]);
    return 0;
}
```

- Blocs/threads 1D
- Chaque thread met à jour 1 élément avec  $k$  multiplications
- $dA[i]$  est accédé  $k$  fois. Peut-on faire mieux?
  - Mettre  $dA[i]$  dans un registre (i.e., `float temp = dA[i];`)
  - Utiliser la mémoire partagée



# Exemple: Calcul de puissance dans un tableau

Étant donné un tableau  $A[N]$  et un entier  $k$ , mettre  $A[i]$  au  $A[i]^k$ .

```
#include <stdio>
#include "cuda.h"

#define N 1024
#define BLOCKSIZE 128
float A[N];

__device__ float dA[N];

__global__ void powerArray(int n, int k)
{
    int i = threadIdx.x + blockIdx.x * blockDim.x;
    // BLOCKSIZE == blockDim.x
    __shared__ float data[BLOCKSIZE];
    if (i < n) {
        data[threadIdx.x] = dA[i];
        float c = 1.0;
        for (int j = 0; j < k; j++) {
            c *= data[threadIdx.x];
        }
        dA[i] = c;
    }
}

int main(int argc, char **argv)
{
    /// ...
    powerArray<<<numBlocks, blockSize>>>>(N, 4);
    /// ...
    printf("%lf\n", A[2]);
    return 0;
}
```

- Le préfixe **\_\_shared\_\_** pour définir un tableau en shared memory
- La taille doit être une **constante** connue en temps de compilation (donc impossible d'utiliser **blockDim.x**)
- $dA[i]$  est accédé **1 fois** puis réutilisé **k fois** en shared memory.
- Tableau est libéré quand le bloc termine donc pas besoin de **désallouer**.
- Utiliser **\_\_syncthreads()** pour attendre tous les threads dans le bloc, avant d'utiliser les données chargées.

- 1 Architecture mémoire d'un GPU
- 2 Utilisation de la mémoire partagée
- 3 Divergence

# Branchements dans un kernel GPU

Les threads sont exécutés par des groupes de 32 dans un **warp**.

- Tous les threads exécute la même instruction simultanément.
- S'il y a un **branchement**: `if (cond) f(); else g();`
  - Si tous les threads dans un warp satisfont **cond**, ils exécutent seulement `f()` simultanément.
  - Si tous les threads dans un warp échouent **cond**, ils exécutent seulement `g()` simultanément.
  - S'il y a **au moins un thread** qui satisfait **cond** et **au moins un threads** qui échoue **cond**
    - D'abord `f()` est exécutée par ceux qui satisfont **cond**, le reste s'endort.
    - Après `g()` est exécutée par ceux qui échouent **cond**, le reste s'endort.
    - Le temps total d'exécution du warp est donc la somme des deux.

# Exemple: Deux types de divergences

Étant donné un tableau  $A[N]$  et un entier  $k$ , mettre  $A[i]$  au  $A[i]^k$ .

```
--device-- void kernel()
{
    // ...
    // Divergence 1
    if (threadIdx.x % 2 == 0) {
        f();
    } else {
        g();
    }
    // ...
    // Divergence 2
    if (threadIdx.x < SIZE / 2) {
        f();
    } else {
        g();
    }
    // ...
}
```

- Supposons  $f()$  et  $g()$  prennent  $F$  et  $G$  secondes respectivement dans l'exécution par un thread.
- Divergence 1 concerne **tous les warps**, donc le temps total d'exécution de chaque warp sera  $F + G$ .
- Divergence 2 concerne **un seul warp** parmi tous, chaque autre warp prend soit  $F$  soit  $G$  secondes dans l'exécution.

# Exemple: Deux types de divergences

Étant donné un tableau  $A[N]$  et un entier  $k$ , mettre  $A[i]$  au  $A[i]^k$ .

```
__device__ void kernel()
{
    // ...
    // Divergence 1
    if (threadIdx.x % 2 == 0) {
        f();
    } else {
        g();
    }
    // ...
    // Divergence 2
    if (threadIdx.x < SIZE / 2) {
        f();
    } else {
        g();
    }
    // ...
}
```

- Pour de bonnes performances, il faut
  - soit éviter les branchements au maximum
  - soit les réaliser de manière que peu de warps divergent dans l'exécution.

## **Contact**

Oguz Kaya

Université Paris-Saclay and LRI, Paris, France

[oguz.kaya@lri.com](mailto:oguz.kaya@lri.com)

[www.oguzkaya.com](http://www.oguzkaya.com)