Devoir 1
INFO4303 - Programmation massivement parallèle

Remise : Vendredi 13 février.

Prof. Andy Couturier

Hiver 2026

Instructions pour le livrable

Le devoir doit être remis via un répertoire Git qui doit être synchronisé sur le serveur git.couturier.prof avant la date et heure de remise.

- Sur le serveur Git, votre répertoire devra être exactement « `INFO4303-H2026-Devoir1` ».
- Les fichiers devront être nommés comme suit :
  - `ex1.cu`
  - `ex1.txt` (sortie du programme)
  - `ex2.cu`
  - `ex2.txt` (sortie du programme)
  - `ex3.cu`
  - `ex3.txt` (sortie du programme)
  - `ex4.cu`
  - `ex4.txt` (sortie du programme)
- Chaque code source doit être écrit en CUDA C/C++. Aucun autre langage de programmation ne sera accepté.
- Chaque code source doit être soigneusement commenté. L'apparence du code doit être propre, avec une bonne indentation.
- Chaque fichier doit être compilable avec le compilateur `nvcc` version 12.x avec la commande suivante sans erreur fatale :
  `nvcc -o exN exN.cu`

  Assurez-vous de remplacer `exN` par le nom de votre fichier source.

Exercice 1

Écrivez un programme CUDA `ex1.cu` qui lance un kernel affichant les informations de chaque thread.

- **Paramètres de ligne de commande :**
  - Le programme doit accepter 4 paramètres : `gridDimX`, `gridDimY`, `blockDimX`, `blockDimY`.
  - Exemple d'utilisation : `./ex1 2 2 4 4` lance une grille 2×2 de blocs 4×4.
- **Kernel :**
  - Pour chaque thread, affichez avec `printf` :
    - `blockIdx.x`, `blockIdx.y`
    - `threadIdx.x`, `threadIdx.y`
    - Un indice global unique calculé (de 0 à N-1, où N est le nombre total de threads)
- **Sortie :**
  - Le format de sortie doit être :

```
Block(0,0) Thread(0,0) -> Global: 0
Block(0,0) Thread(1,0) -> Global: 1
...
```

- **Test requis :**
  - Exécutez `./ex1 2 1 4 2` et enregistrez la sortie dans `ex1.txt`.

Exercice 2

Écrivez un programme CUDA `ex2.cu` qui remplit une matrice avec les indices calculés par les threads.

- **Paramètres de ligne de commande :**
  - Le programme doit accepter 4 paramètres : `rows`, `cols`, `blockDimX`, `blockDimY`.
  - Exemple d'utilisation : `./ex2 16 16 8 8` crée une matrice 16×16 avec des blocs 8×8.
- **Fonctionnement :**
  - Allouez une matrice de dimensions `rows` × `cols` sur le GPU.
  - Chaque thread calcule sa position `(row, col)` et écrit la valeur `row * cols + col` à cette position.
  - Gérez correctement le cas où les dimensions de la matrice ne sont pas des multiples de la taille des blocs.
- **Sortie :**
  - Copiez le résultat sur le CPU et affichez la matrice complète.
  - Affichez également :

    - Les dimensions de la grille (nombre de blocs en X et Y)
    - Le nombre total de threads lancés
    - Le nombre de threads « hors matrice » (qui ne traitent aucune donnée)
- **Test requis :**
    - Exécutez `./ex2 10 12 4 4` et enregistrez la sortie dans `ex2.txt`.

Exercice 3

Écrivez un programme CUDA `ex3.cu` qui double chaque élément d'un vecteur et compare différentes configurations de blocs.

- **Fonctionnement :**
    - Créez un vecteur de 1 000 000 d'éléments initialisés à 1.0.
    - Le kernel doit doubler chaque élément : `data[i] = data[i] * 2`.
    - Le programme doit tester automatiquement les tailles de bloc suivantes : 32, 64, 128, 256, 512, 1024.
- **Mesure du temps :**
    - Utilisez `cudaEvent_t` pour mesurer le temps d'exécution du kernel.
    - Ne mesurez que le temps du kernel, pas les transferts mémoire.
- **Sortie :**
    - Pour chaque configuration, affichez :
        - Taille de bloc
        - Nombre de blocs
        - Nombre total de threads lancés
        - Nombre de threads inutiles (qui ne traitent aucune donnée)
        - Temps d'exécution en millisecondes
    - Vérifiez que le résultat est correct pour chaque configuration.
- **Test requis :**
    - Exécutez `./ex3` et enregistrez la sortie dans `ex3.txt`.

Exercice 4

Écrivez un programme CUDA `ex4.cu` qui inverse les couleurs d'une image en niveaux de gris en utilisant une configuration 2D.

- **Paramètres de ligne de commande :**
    - Le programme doit accepter 2 paramètres : `width` et `height`.
    - Exemple d'utilisation : `./ex4 1920 1080`.
- **Fonctionnement :**
    - Créez une image de test où chaque pixel a la valeur `(x + y) % 256`.
    - Utilisez une configuration 2D avec des blocs de 16×16 threads.

  - Le kernel applique l'inversion : `pixel = 255 - pixel`.
  - Gérez correctement les dimensions qui ne sont pas des multiples de 16.
- **Vérification :**
  - Après l'exécution du kernel, vérifiez le résultat en comparant avec un calcul effectué sur le CPU.
  - Affichez « Vérification réussie » ou « Erreur de vérification » selon le cas.
- **Sortie :**
  - Affichez :
    - Les dimensions de la grille (blocs en X et Y)
    - Le nombre total de threads lancés
    - Le nombre de threads hors image
    - Le résultat de la vérification
- **Test requis :**
  - Exécutez `./ex4 1000 500` et enregistrez la sortie dans `ex4.txt`.