Programmation GPU 2

Rattrapage

EPITA - SSIE - S9

Juin 2024

Accéder à un GPU

Pour accéder en `ssh` à un GPU : `ssh -l LOGIN -p PORT gpgpu.image.lrde.iaas.epita.fr`, où `LOGIN` est sous la forme `prenom.nom`, et `PORT` est un nombre compris entre `22000` and `22004`. Favoriser l'utilisation d'un répertoire Git car les fichiers hors de l'AFS ne sont pas persistants.

Figure fractale de l'ensemble de Julia

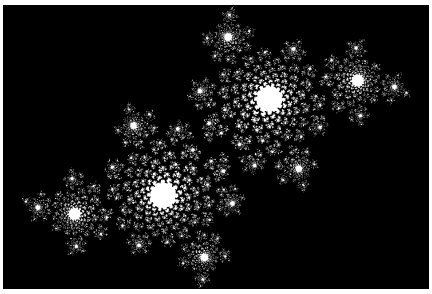

Figure 1: Image binaire.

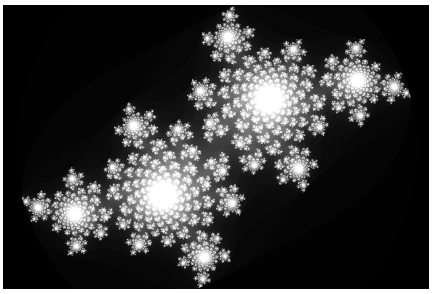

Figure 2: Image en niveau de gris.

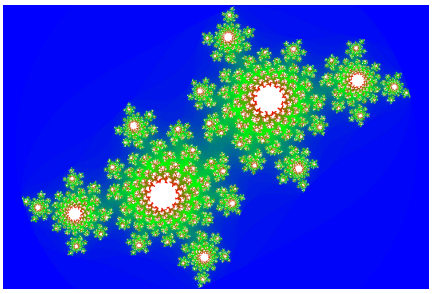

Figure 3: Image colorée.

L'ensemble de Julia est défini par la suite

$$z_{n+1} = z_n^2 + c \tag{1}$$

où z_n et c sont des nombre complexes. Le nombre c est considéré comme constant et vaut ici $c = -0.5 + 0.6i$. La suite est initialisée à z_0 qui correspond aux coordonnées d'un pixel d'une image où la partie réelle correspond à l'axe x et la partie imaginaire à l'axe y. L'objectif est de générer une image de N colonnes et M lignes où chaque pixel d'indice (i, j) indique si la suite de l'Equation 1 converge ou pas. On considère que le suite converge dés lors que $|z_n|^2 < 2$, $\forall n \leq 100$, où $|z_n|$ est le module de z_n. On se concentre sur la zone $x \in (-1.5, +1.5)$ en horizontal et $y \in (-1.0, +1.0)$ en vertical. On note C le nombre de canaux dans l'image générée, valant 1 dans les deux premières étapes, et 3 (pour les composantes rouge, vert et bleu) dans la troisième et dernière étape.

Le fichier `exercice.h` contient la fonction `get_ptr` pour accéder au contenu de l'image. Dans tous les exercices, la mémoire doit être allouée sur GPU en utilisant la fonction `cudaMallocPitch`. Les noyaux CUDA 2D sont tous lancés avec `BLOCK_SIZE`×`BLOCK_SIZE` *threads* par *block*.

Rendu

L'objectif est d'implémenter les 3 exercices suivants. Il faudra rendre une archive `.zip` qui contient les sources et le `Makefile` (sans executables ni images). La notation prend en compte le respect des consignes, la qualité du code, et la bonne utilisation des fonctionnalités de CUDA.

Exercice 1. Image binaire (4/10 points)

Le première étape consiste à générer une image en noir et blanc où un pixel est blanc (= 1.0) si la suite converge, ou noir (= 0.0) sinon.

1. Dans le fichier `exercice.cu`, compléter le noyau CUDA 2D `kernel_generate1` qui traite un pixel de l'image binaire générée avec $C = 1$.
2. Dans le fichier `exercice.cu`, compléter la fonction `generate1` pour allouer de la mémoire sur GPU, lancer le noyau CUDA avec `BLOCK_SIZE`×`BLOCK_SIZE` *threads* par *block*, rapatrier les données sur CPU, libérer la mémoire allouée sur GPU, puis retourner le résultat.
3. Compiler le fichier `main1.cu` avec la commande `make main1` puis exécuter `./main1` et comparer le résultat avec la Figure 1 qui se trouve aussi dans le fichier `ref1.jpg`.

Exercice 2. Image en niveau de gris (3/10 points)

La deuxième étape consiste à générer une image en niveau de gris où un pixel est blanc (= 1.0) si la suite converge, ou gris sinon. En cas de divergence, la valeur du gris (entre 0.0 et 1.0) dépend du ratio entre l'itération n où la suite commence à diverger et le nombre maximal d'itérations (100).

1. Dans le fichier `exercice.cu`, compléter le noyau CUDA 2D `kernel_generate2` qui traite un pixel de l'image en niveau de gris générée avec $C = 1$.
2. Dans le fichier `exercice.cu`, compléter la fonction `generate2` similairement à `generate1`.
3. Compiler le fichier `main2.cu` avec la commande `make main2` puis exécuter `./main2` et comparer le résultat avec la Figure 2 qui se trouve aussi dans le fichier `ref2.jpg`.

Exercice 3. Image colorée (3/10 points)

La dernière étape consiste à générer une image colorée où un pixel est blanc (= 1.0) si la suite converge, ou correspond à un mélange de bleu, vert et rouge sinon. En cas de divergence, la couleur dépend dépend du ratio. La couleur est bleue (0.0, 0.0, 1.0) si le ratio vaut 0.0, verte (0.0, 1.0, 0.0) si le ratio vaut 0.5, rouge (1.0, 0.0, 0.0) si le ratio tend vers 1.0. Entre ces valeurs, les couleurs sont interpolées linéairement. Par exemple, si le ratio vaut 0.75 alors la couleur sera entre le vert et le rouge, soit jaune foncé (0.5, 0.5, 0.0).

1. Dans le fichier `exercice.cu`, compléter le noyau CUDA 2D `kernel_generate3` qui traite un pixel de l'image de couleur générée avec $C = 3$.
2. Dans le fichier `exercice.cu`, compléter la fonction `generate3` similairement à `generate2`.
3. Compiler le fichier `main3.cu` avec la commande `make main3` puis exécuter `./main3` et comparer le résultat avec la Figure 3 qui se trouve aussi dans le fichier `ref3.jpg`.