PPAR: TP GPU 2

Caroline Collange et Antonio Mucherino

Mars 2021

L'objectif de ce TP est de réaliser une version à 2 joueurs du Jeu de la vie de Conway [1].

Nous considérons un domaine en 2 dimensions composée de `domain_x` × `domain_y` cellule. Chaque cellule peut être soit rouge, bleue, ou vide. Le domaine a une forme torique : la voisine de droite de la cellule la plus à droite correspond à la cellule la plus à gauche, et inversement, et de même verticalement.

Chaque cellule a 8 voisines dans les cellules adjacentes. À chaque pas de temps, toutes les cellules évoluent selon les règles suivantes :

— une cellule qui a strictement moins de 2 voisines en vie parmi les 8 cellules adjacentes meurt (devient vide),
— une cellule qui a strictement plus de 3 voisines en vie meurt également,
— une cellule en vie qui a 2 ou 3 voisines en vie survit,
— une cellule vide qui a exactement 3 voisines en vie naît (devient occupée). Sa couleur est celle de la majorité de ses voisines (c'est-à-dire si 2 cellules voisines ou plus sont bleues, la nouvelle cellule est bleue, et rouge sinon).

Toutes les cellules sont mises à jour en même temps, indépendamment.

Comme dans le TP précédent, vous pouvez vous connecter à *parawell* par la commande :

```
ssh -i pparXX_id_rsa pparXX@parawell.irisa.fr
```

où `pparXX` est votre identifiant et `pparXX_id_rsa` est le nom de fichier de votre clé, chemin d'accès inclus. Une fois connecté·e à votre compte sur *parawell*, vous partirez du squelette de code dans le répertoire : `gpu_lab2`.

Pour éviter des conditions de course entre threads, nous suivons une approche *ping-pong* : on maintient deux copies du domaine. À chaque pas de temps, on lit l'ancien état des cellules depuis une copie en écrivant le nouvel état dans l'autre copie, puis on échange les pointeurs des deux domaines. Ainsi, chaque copie sert alternativement de source et de destination.

On représente chaque cellule du domaine par un entier : 0 pour une cellule vide, 1 pour rouge, 2 pour bleue.

1. Programmer la simulation sans chercher à optimiser les accès mémoire. On utilisera la fonction `read_cell` pour accéder aux cellules voisines.

1 Optimiser les accès mémoire

2. Combien d'accès mémoire en lecture depuis la mémoire globale effectue chaque thread ? Combien d'accès en lecture cela fait-il pour un bloc de threads CUDA ?

On veut maintenant réduire le nombre d'accès à la mémoire globale. Pour ce faire, on va partager et réutiliser les données entre les threads de chaque bloc CUDA.

3. Considérons un bloc de threads. Combien de cases en mémoire globale sont-elles lues par au moins un thread du block ? Contrairement à la question précédente, les cases qui sont lues par différents threads du même bloc ne sont comptées qu'une seule fois. *Vous pouvez faire un dessin*

4. Même question si les blocs ont 2 dimensions. Quelle forme de bloc minimise le nombre de cases mémoires uniques à lire ?

5. Modifier le code pour éviter les lectures redondantes en mémoire globale, en utilisant la mémoire partagée.

Références

[1] https://fr.wikipedia.org/wiki/Jeu_de_la_vie