

Simulation de particules sur architectures parallèles - version OpenCL

Alexis Couture, Marc Sarraute

3 mai 2015

1 Bilan

Pour ce second livrable, nous avons implémenté une étape du tri par boîte, la fonction scan, sous forme de noyau OpenCL (le reste du tri étant déjà implémenté).

On a réussi à implémenter un scan (ou prefix sum) qui s'exécute de manière parallèle. On a en entrée un tableau `box_buffer` contenant le nombre d'atomes par boîtes et on écrit le résultat dans `calc_offset_buffer`. Notre fonction se déroule de la manière suivante :

- On divise `box_buffer` en le stockant par parties dans la mémoire locale de chaque workgroup, dans un tableau `temp`
- Chaque workgroup réalise un prefix sum sur sa partie de `box_buffer`, toujours sur `temp`
- Chaque workgroup stocke la dernière valeur de son prefix sum dans `box_buffer` (le workgroup `i` stocke son résultat dans la case `i`)
- Chaque workgroup fait un prefix sum sur `box_buffer`, et le stocke en mémoire locale dans le tableau `local_sumz`
- Chaque workgroup ajoute à son prefix sum, la somme des prefix sum obtenus par les workgroups précédents (traitant les parties antérieures de `box_buffer`)
- On écrit les valeurs stockées dans `temp` dans `calc_offset_buffer`

Notre noyau utilise un thread par boîte et partage le travail entre plusieurs workgroups. On utilise le tiling pour réduire, mais sans éliminer, le problème de dépendance inhérent à l'opération scan, où chaque thread a besoin du résultat du thread travaillant sur la case précédente. Au final, nous obtenons de moins bonnes performances qu'avec le tri par Z, signe que notre noyau est assez perfectible.

2 Limitations

Tout d'abord, il n'existe pas en OpenCL de moyen propre pour synchroniser les workgroups entre eux car ceux-ci doivent travailler de manière indépendante. Pourtant, chaque workgroup a besoin des résultats des workgroups précédent pour terminer son calcul. Pour traiter ce problème, chaque workgroup stocke son résultat en mémoire globale dans `box_buffer`, à la case d'indice de son `group_id`, et récupère le résultat des autres workgroups sans vérifier si ceux-ci ont fini leur calcul.

Ensuite, notre implémentation a une complexité linéaire $n + m$ (où n est le nombre de boîtes et m le nombre de workgroups) et utilise n threads (un thread par boîte), ce qui n'est pas efficient. En conséquence, nous n'obtenons pas de performances supérieures à celles du tri par Z version OpenCL, ce qui n'est pas satisfaisant étant donné que le tri par boîte est théoriquement plus rapide. Une idée d'amélioration serait de

suivre l'implémentation décrite dans ce papier (LE PDF DE M.HARRIS), qui utilise moins de threads et possède une complexité logarithmique.

3 Tests