pap-rendu-2

Tarrieu Odrian et Drezen Corentin

March 2024

1 Introduction

2 [4.3] OpenMP implementation of the asynchronous version

Pour que notre implémentation de asandPile_compute_omp puisse fonctionner, il nous a aussi fallu modifier le fonction do_tile de la version asynchrone. En effet, le do_tile ne garantissait pas la synchronisation des écritures en bord de tuiles.

Ainsi, nous avons implémenter une fonction inline pour gérer les bords de tuiles et ne pas avoir à écrire toutes les 'if' imbriqués (et rendre le code plus lisible).

On a aussi utilisé des pointeurs vers les cellules pour pouvoir faire fonctionner cela.

On a aussi essayé d'optimiser le calcul des tuile (do_tile) de rajouté un pragma pour demander à gcc de dérouler les boucle mais cela semblait ne rien changer voir empirer les performances avec "unroll-all-loops".

De plus on a essayé d'implémenter une fonction first touch mais la répartition des threads sur le cache ne doit pas être bon. On semble gagner moins de 5 ms mais nous ne sommes pas parvenu à faire une expérience qui le démontre.

Pour la parallélisation de la version asynchrone (asandPile), nous avons réalisé les tests sur des tuiles carrées, linéaires et rectangulaires.

Les résultats des expériences avec des tuiles carrées et linéaires sont dans la figure 1, et ceux avec des tuiles rectangulaires sont dans la figure 2.

Nous avons observé que les meilleurs temps de calcul étaient obtenus avec des tuiles rectangulaires de dimensions 64x16 avec 16 threads et l'ordonnancement static. On a aussi testé la version dynamique mais elle semblait trop lente pour l'inclure dans l'expérience finale.

On a une trace qui montre que l'on perd du temps peut être à cause la barrière implicite entre les deux "for collapse(2)"

Après des experience sur des taille d'image de surface 2048x2048 on a réalisé qu'il y avait un bug, sans doute qu'il manque une synchronisation ou que la synchronisation des bords de tuile est mal faite. Le bug semble plus courant

avec des tuiles rectangulaires de la plus petite hauteur possible et sans option (spirals ou alea).

Sur la Figure 6 on peut voir que certaine expériences on un nombre d'itérations beaucoup plus petit et donc un temps plus court (surtout en static,1) mais un résultat faux.

On a finalement réalisé des expériences pour les image plus grande qui semblent écarter les bugs qui ont l'air présent avec des tuiles de hauteur inférieur à 8. (cf Firgure 7 & 8)

Figure 1: Expérience 4.3, asandPile_omp avec tuiles carrées et linéaires

3 [4.4] Lazy OpenMP implementations

Nous n'avons pas réussi à implémenter les versions lazy des fonctions synchrones et asynchrones. Le résultat que nous obtenons est représenté sur la figure 4. On constate qu'il nous manque au moins de tuiles qui devrait ne pas être considérés stables.

On pensé qu'il devait rester des tuiles instable à la fin des itérations. On a remarqué que si on rajoute une itération après "change == 0" on obtient un résultat plus satisfaisant mais avec un sha256 différent (cf figure 5)

On a passé la majeure partie de notre temps sur la version synchrone lazy, et nous ne comprenons pas exactement pourquoi elle ne fonctionne pas.

Même si nous n'avons pas réussi notre fonction paresseuse pour la version synchrone comme pour la version asynchrone on a tout de même réaliser une expérience pour trouver le meilleur temps avec l'option "spirals" et sur une surface de 1024x1024 pour la version synchrone optimisée au premier rendu. (cf Figure 9)

On observe les mêmes chutes de performances avec certaines valeurs de OMP_NUM_THREADS que pour une experience similaire du rendu 1.

Figure 2: Expérience 4.3, asandPile_omp avec tuiles rectangulaires

Figure 3: Expérience 4.3, traces pour asandPile_compute_omp 64x16 static

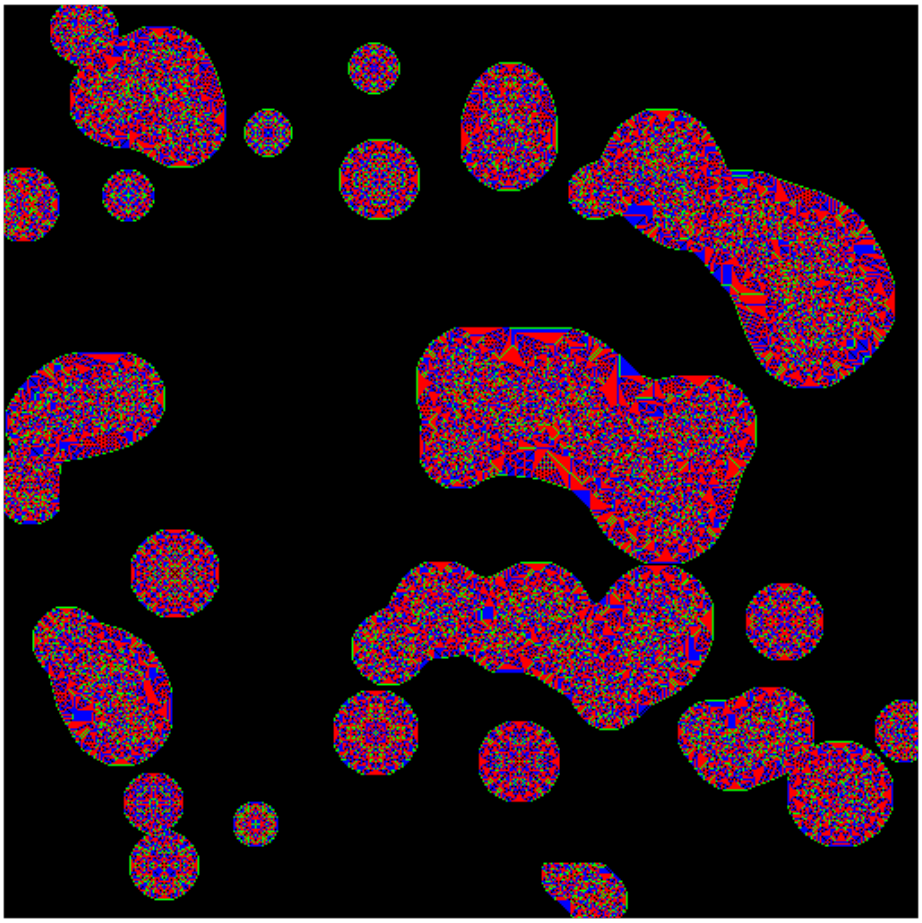

Figure 4: Expérience 4.4, résultat de notre implémentation lazy synchrone

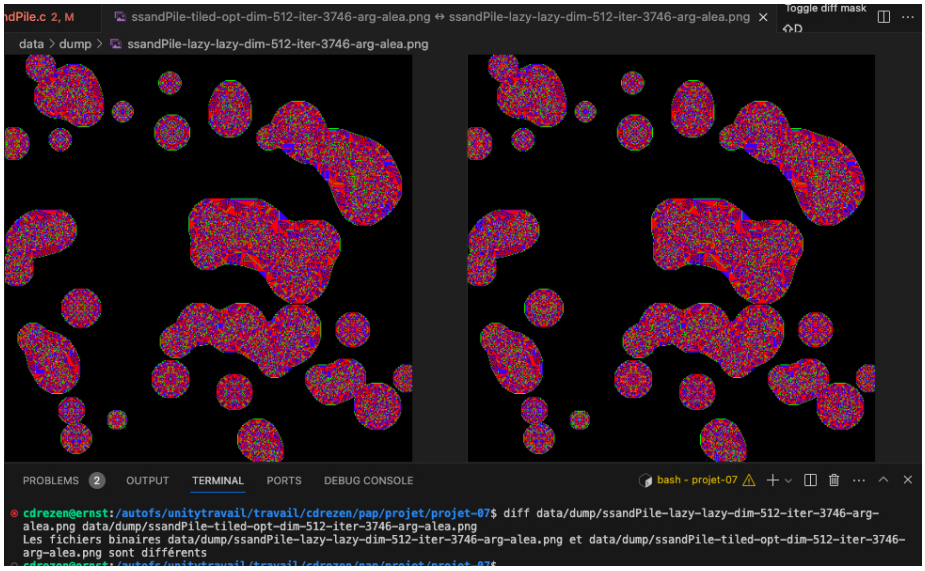

Figure 5: Expérience 4.4, résultat de notre implémentation lazy synchrone avec double itérations

Figure 6: Expérience 4.3, Mise en évidence du bug

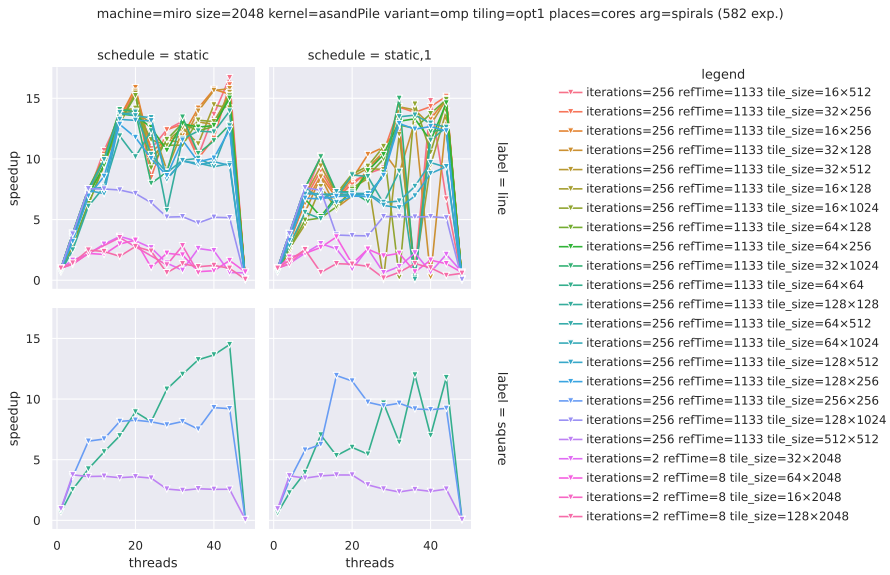

Figure 7: Expérience 4.3, perf asandPile 2048 spirals (sans lazy)

Figure 8: Expérience 4.3, perf asandPile 8192 alea (sans lazy)

Figure 9: Expérience 4.2, perf ssandPile 1024 spirals (sans lazy)