V03 TP1

Téo Boutin

20 Octobre 2020

1 Grain Fin

Pour la parallèlisation grain fin, j'ai juste ajouté la ligne :

#pragma omp parallel **for default**(shared) **private**(du1, du2, du, x, y, z, i, j, k) reduction(+: du_sum)

juste avant la boucle spatiale, dans la fonction Scheme::iteration_domaine du fichier scheme.cxx.
Les variables impliquées doivent presque toutes être définies comme privées.
En termes de performances, l'acceleration stagne à 3 à partir de 5 threads.

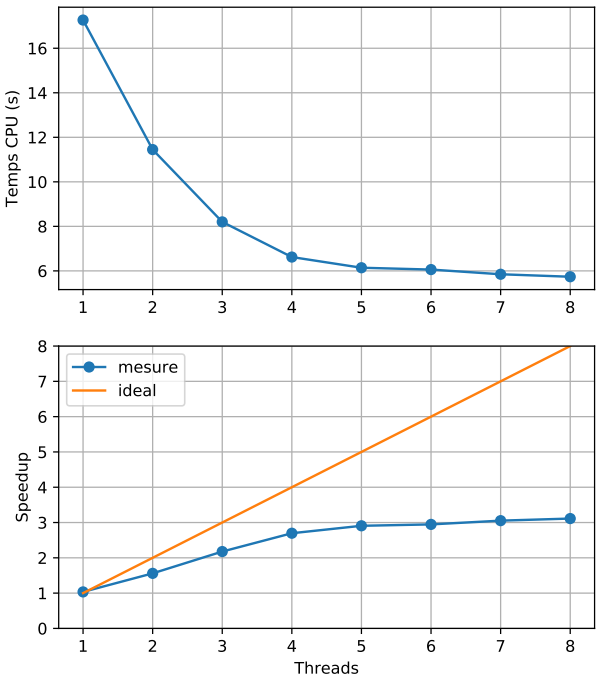

Figure 1: Performance de la parallelisation grain fin

2 Grain grossier

Pour la parallèlisation grain grossier, il y a plus de code à ajouter.
On commence par ouvrir une région parallèle autour de la boucle en temps. Les variables créées dans la région parallèle sont privées de base, donc il n'y a pas à les spécifier au début. Dans cette boucle, la seule partie qui doit être effectué par plusieurs threads est l'itération (la boucle spatiale). Pour les autres parties, j'ai utilisé des **#pragma** omp master et une barrière après

l'itération car les affichages de résultats étaient incorrects autrement.
Dans C.iteration(), on fait executer à chaque thread une partie des itérations spatiales, en découpant la boucle manuellement. Après la boucle, on fait la reduction des variations avec un critical , et il faut ensuite faire l'échange des nouveau et ancien vecteurs. J'ai mis une barrière avant car sinon l'échange se faisait parfois avant qu'un des threads ait fini sa boucle spatiale. Un seul des threads doit faire cet échange.

Les performances ne sont pas meilleures que pour la parallèlisation grain fin, sans doute à cause des nombreuses barrières qui se trouvent dans mon code.

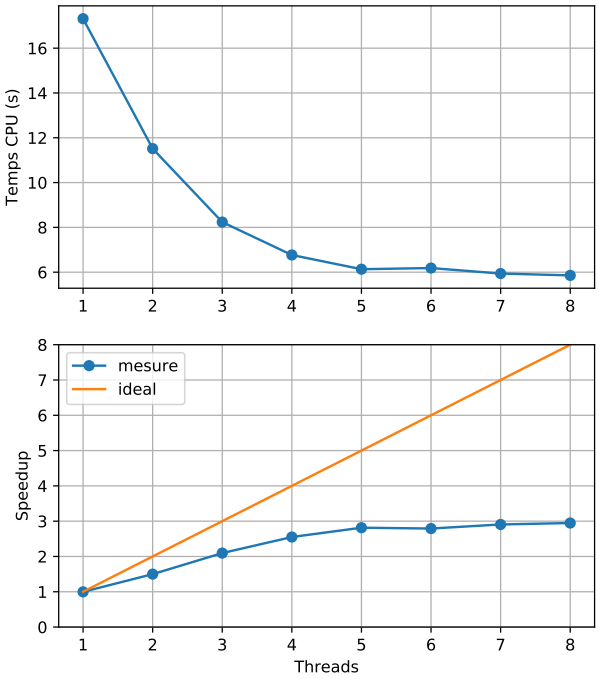

Figure 2: Performance de la parallelisation grain grossier