AMSI03 - TP1

Quentin Goepfert

1 Introduction

On cherche à mettre en pratique l'utilisation de OpenMP sur un code de calcul 3D de l'équation de Poisson par différences finies. On possède déjà un code fonctionnel que l'on veut parallélisé. On s'intéréssera à deux méthodes : une méthode "grain fin" et une méthode "grain grossier".

2 Parallélisation

Pour paralléliser le code, on doit évidemment identifier quelles opérations peuvent être faites en parallèle. On remarque que les itérations en temps sont dépendantes les unes des autres (on a besoin de la valeur au temps t pour avoir la valeur au temps $t + dt$). Pour le calcul de la valeur de notre solution u à un instant donné t en un point (x,y,z), on à seulement besoin de la valeur des voisins de ce point au temps $t - dt$. Ainsi, si on suppose que tous les coeurs connaissent la valeur de u au temps $t - dt$, le code peut être parallélisé en espace.

Le calcul de la solution en un temps donné dans tout le volume se traduit par une boucle for. Pour la paralléliser, on va devoir identifier quelles sont les variables locales (privées) et lesquelles sont des variables globales (partagées), qui ne seront pas modifiées par un coeur.

3 Grain Fin

Pour commencer la parallélisation de la boucle, on décide de n'utiliser que des *#pragma* simples autour des boucles. On identifie la boucle sur les sommets de notre maillage dans la méthode *iteration_domaine*() de la classe *Scheme* et on va classer toutes les variables. On ne touche pas à la méthode *iteration*() pour l'instant, car on ne découpe pas le domaine entre chaque coeur. Les variables qui doivent être privés sont :

- x,y,z : Le point courant dépend du coeur
- $du1, du2, du$: Il s'agit de la valeur de la fonction calculée en un point par un certain processeur.
- *du_sum* : On doit faire la somme des du, ainsi on fait une réduction par la somme.

Les autres variables sont partagées entre les processeurs car on n'accède/ne modifie pas une variable par au moins deux coeurs simultanément.

Ainsi, on rajoute le *#pragma* suivant avant les boucles sur les noeuds :
 # pragma omp parallel for private (x, y, z, du1, du2, du)
shared (xmin, ymin, zmin, m_dx, m_dt, m_f, m_u)
reduction (+ : du_sum) collapse (3)

4 Grain Grossier

Le problème de la parallélisation est que l'on créé des espaces parallèle à chaque itération de temps qui sont détruites ensuite. Pour éviter ce problème, on va créer une zone parallèle autour de l'itération au temps, et activer les threads lorsqu'il le faut. Pour simplifier le code, on va découper le domaine et assigner les sommets entre les coeurs. Ainsi, le découpage de la boucle sur les sommets est déjà fait en amont, et on a plus besoin de répartir les variables partagées et privées. On doit faire attention à la variables *du_sum*. Désormais, la méthode *iteration_domaine*() renvoie une variable local, que l'on doit réduire ensuite par la somme.