Rapport I03 : TP 1

Gabriel Hadjerci

1 Grain fin

La version multithreads avec OpenMP en grain fin consiste à ouvrir des régions parallèles à chaque fois qu'une parallélisation est possible.

Ici la discrétisation en temps n'est pas parallélisable puisqu'on a besoin des valeurs de la solution à l'instant t pour calculer la solution à l'instant $t + dt$. Ainsi ce qu'on peut paralléliser ce sont les boucles sur l'espace comme la boucle sur les noeuds dans la fonction *iteration_domaine*. L'intérêt du grain fin c'est que la mise en place est simple, une seule ligne de code est nécessaire pour paralléliser plusieurs boucles. Par exemple, pour la boucle de *iteration_domaine* on peut écrire simplement :

 # pragma omp parallel for private(du1,du2,x,y,z,du) reduction(+ : du_sum) collapse(3)

avant les boucles *for*. Les intermédiaires de calculs sont passés en privés afin d'éviter les problèmes de mémoires. De même, les indices sont implicitement privés. À la fin de la boucle, on effectue une réduction sur le terme *du_sum* car il s'agit initialement d'une somme sur tout l'espace. Enfin le terme *collapse*(3) permet de paralléliser les boucles sur les trois directions de l'espace en même temps.

Pour finir, on peut paralléliser les boucles sur l'espace dans *values.cxx* qui servent à l'initialisation.

2 Grain grossier

La version grain grossier consiste à n'ouvrir qu'une seule région parallèle afin d'éviter les coûts non nécessaires. Dans notre cas on ouvre la région parallèle juste avant la boucle temporelle. On pourrait aussi paralléliser l'initialisation, cependant c'est un peu compliqué d'initialiser des objets sur plusieurs threads et faire en sorte que tous y ait accès. J'ai donc uniquement parallélisé la boucle spatiale qui s'exécute à chaque itérations.

La quasi-intégralité de la boucle temporelle se fait sur un seul thread, seul la partie *C.iteration(int iThread)* qui se trouve dans *scheme.cxx* s'exécute en parallèle. Ce qui change par rapport à la version séquentielle, c'est qu'on passe le numéro du thread *iThread* en argument. Cela permet d'éviter de le rechercher à chaque fois. On utilise ensuite les fonction *imin_local* et *imax_local* pour obtenir les bornes des

trois boucles sur l'espace. Ainsi, il n'y a pas à modifier *iteration_domaine*, il faut seulement passer en argument les bonnes bornes.

La partie compliquée intervient lorsqu'il faut réduire le résultat de la fonction *iteration_domaine*. Pour cela j'ai choisi de rentrer dans un tableau le résultat pour chaque thread, puis sur un seul thread d'effectuer la somme de toutes les valeurs du tableau. Cela nécessite de mettre une barrière avant la somme afin d'être sûr que chaque thread ait bien eu le temps de calculer sa partie.

3 Conclusion

Finalement, avec ces deux méthodes on arrive a accélérer l'algorithme bien que l'accélération semble bornée. De plus les deux méthodes sont plus ou moins équivalentes sur cet exemple puisqu'on a à peu près les mêmes résultats.