Rapport I03 : TP 2

Gabriel Hadjerci

1 Grain fin

La version hybride en grain fin consiste à ouvrir des régions parallèles OpenMP uniquement lorsqu'une boucle est parallélisable. Les communications MPI se font alors en dehors des régions parallèles OpenMP. Cela implique d'utiliser le niveau de compatibilité **MPI_FUNNELED**.

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. Ici je n'ai paralléliser que 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(j,k,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, l'indice i est implicitement privé. 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.

La réduction MPI est faite juste après cette boucle sur le thread principal.

2 Grain grossier

La version grain grossier consiste à n'ouvrir qu'une seule région parallèle OpenMP afin d'éviter les coûts non nécessaires. Dans notre cas on ouvre la région parallèle juste avant la boucle temporelle car j'ai choisi de ne pas paralléliser l'initialisation. En ce qui concerne les communications MPI, elles ne se feront que dans les portions **omp single**, on utilise donc le niveau de comptabilité **MPI_SERIALIZED**.

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_thread* et *imax_thread* pour obtenir les bornes

des trois boucles sur chaque domaine MPI. Le découpage des domaines MPI est fait dans *parameters.cxx*. Il est effectué de sorte que ce soit la dimension avec le plus de points qui soit découpée.

La partie compliquée intervient lorsqu'il faut réduire le résultat de la fonction *iteration_domaine*. Pour cela, j'ai choisi de faire la réduction OpenMP en premier à l'aide d'une portion **omp atomic** :

```
#pragma omp atomic
m_temp_global += m_temp_local;
```

où la variable *m_temp_local* est le résultat de la fonction *iteration_domaine* de laquelle on a retiré la réduction MPI. On effectue la réduction MPI juste après dans une zone **omp single**. Cela nécessite de mettre une barrière OpenMP après la réduction OpenMP afin d'être sûr que chaque thread ait bien eu le temps d'ajouter sa part du travail avant de commencer la réduction MPI.

3 Conclusion

Finalement, les deux méthodes sont fonctionnelles et fournissent les bons résultats. Malheureusement, on ne voit pas forcément d'accélération par rapport au tout MPI car le nombre de processus MPI est trop petit. Tout de même on constate que la version grain fin est plus rapide en moyenne que la version grain grossier. Cela peut être dû à la réduction avec *pragma omp atomic*. J'ai voulu essayer de cette manière mais c'est peut être plus lent que d'utiliser un tableau temporaire et de sommer termes à termes.