*Teixeira Quentin*

**Rapport Projet IN203**

Parallélisation d’une simulation d’une colonisation d’une galaxie

Avant tout changement du code, au lancement des programmes, on a un temps de calcul de 20 ms en moyenne et un temps d’affichage de 7 ms en moyenne, soit un temps d’exécution des programmes de 27 ms en moyenne pour une boucle.

1. **Parallélisation de boucle en mémoire partagée**

Pour faire une version Multithread du code, on utilise la bibliothèque OpenMP. On crée des sections dans la boucle ‘while’ du fichier colonisation.cpp.

Résultat : Pour 2 threads, on a toujours 20 ms en moyenne pour le temps calcul, pour le temps d’affichage c’est maintenant plutôt 7.5 ms en moyenne. Cette petite augmentation est peut-être due à la division en section qui prend du temps. Ainsi, pour une boucle compète, cela prend 20 ms en moyenne !

On obtient les mêmes résultats pour 3, 4, et 5 threads, ce qui paraît étonnant vu que ma machine possède seulement 2 threads par cœur. J’imagine que lorsque l’on demande à OpenMP de créer plus de threads qu’il y en a, il en créer des ‘virtuels’, et que tant qu’on ne demande pas d’en créer trop cela ne réduit pas les performances. Autrement les résultats sont moins bons…

L’accélération vaut donc S = 27/20 = 1.35 !

Il n’y a apparemment aucun conflits mémoire ici.

1. **Parallélisation en mémoire distribuée**

Pour faire la version partagée du code, on utilise la bibliothèque MPI. On utilise 4 processus au mieux (c’est le maximum sur l’ordinateur utilisé) ; le premier sera utilisé pour traiter l’affichage, et les trois autres pour les calculs. Ces trois derniers processus se partageront la grille en bandes de largeur égale à celle de la grille d’origine, et de hauteur égale à un tier de celle d’origine (+1 pour les bandes de haut et bas, et +2 si c’est la bande du milieu). Le code est cependant prévu pour fonctionner avec n processus, n > 3.

(Temps en moyenne :)

3 processus : Temps récupération données = 2.5 ms, Temps pour une boucle = 12 ms.

4 processus : Temps récupération données = 0.7 ms, Temps pour une boucle = 11.5 ms.

5 processus : Temps récupération données = 0.6 ms, Temps pour une boucle = 12 ms.

(Pout utiliser plus que 4 processus il faut faire -oversubscribe en plus, et on obtient globalement les mêmes résultats que pour 4, logique puisqu’il y a 4 processus dans ma machine.)

Je pense que le temps de récupération des données (temps du point de vue du processus 0, celui qui reçoit les données pour en faire l’affichage) est court non pas parce que le temps de calcul est très court, mais parce que les calculs sont faits en amont et que les processus attendent le processus 0. Du coup, dès que le processus demande les résultats, ils sont prêts.

En effet, pour 4 processus, ils mettent environs 8 ms en moyenne pour faire les calculs. Donc la tâche la plus longue est l’affichage. Avec 3 processus, le temps de calcul est de 12 ms en moyenne. Comme c’est autant que le temps d’affichage (processus 0), voilà pourquoi on ne voit pas d’amélioration entre 3 et 4 processus…

Finalement, au mieux avec 4 processus, on a une accélération de 27/10.5 = 2.6 !

1. **Parallélisation en mémoire partagée et en mémoire distribuée**

On pourrait tenter de séparer encore en deux threads chacun des processus de calcul pour accélérer les tâches de calculs (autres que processus 0). Je n’y vois toutefois pas d’intérêt puisque c’est uniquement l’affichage qui est la tâche critique avec 4 processus. Il faudrait trouver le moyen de l’accélérer…

J’ai essayé d’utiliser OpenMP sur la fonction render (galaxie.cpp) parce qu’il me semble que c’est celle là qui fait l’affichage, mais sans succès ; Error : Segmentation fault. Je pense que l’on ne peut pas demander à plusieurs threads de modifier l’image en même temps…

**Conclusion**

Ce projet m’a permis de me rendre compte concrètement que OpenMP est assez simple d’utilisation par rapport à MPI, mais qu’utiliser les processus est beaucoup plus efficaces que de simplement séparer les tâches entre les threads. Finalement, il semble que l’amélioration de la vitesse d’exécution des calculs est une chose aisée lorsque l’on maitrise les processus et les threads, mais que pour l’affichage, c’est une autre histoire…