Rapport Projet IN203

David Velasquez Ospina

January 2020

1 Introduction

Ce projet vise à comparer le rendement de différentes approches de programmation parallèle pour un même programme. Le dossier de projet est "Projet_Laby_ants" et est un dossier git avec différentes branches ; chacune avec une approche différente mais avec la graine de hasard fixée à 18 et avec une taille de 32 x 64. Les différentes approches sont : OpenMp, MPI avec 2 processus, MPI avec 12 processus.Au final, on se rend compte qu'en termes de vitesse, la plus rapide avec les méthodes utilisées est le MPI avec traitement 12 fils.

2 OpenMP

Pour l'OpenMP, la visualisation des calculs a été séparée à l'aide de la fonction Sections. Ensuite, la fonction "advance_time" a été parallélisée à l'aide d'une omp for parallel et avec une reduction de la variable "cpteur". Finalement un critical a été ajouté dans la fonction mark_pheromone.

3 MPI 2 Processus

Dans ce cas, une communication point à point a été effectuée entre le processus 0 et le processus 1. Le processus 0 s'occupe de l'affichage et le processus 1 des calculs. Une expédition de blocage a été utilisée avec un tampon contenant la quantité de nourriture, les phéromones et la position des fourmis.

4 MPI 12 Processus

Ce programme peut être exécuté avec un minimum de 3 processus et le maximum est imposé par la machine sur laquelle il est exécuté. Les calculs de temps ont été effectués avec 12 fils de traitement. Le processus 0 est chargé de la visualisation et communique avec le processus 1 ; qui calcule une partie des fourmis, reçoit le calcul des autres processus et envoie le cumul dans un buffer au processus 0. Les autres processus sont chargés de calculer et d'envoyer leurs

résultats au processus 1, à travers un canal de communication différent de celui entre 0 et 1.

5 Résultats

Comme nous pouvons le voir dans le tableau 1, le Speed up le plus élevée est donnée par la parallélisation avec MPI et 12 processus. Suivi par OpenMP, qui a également eu une bonne accélération. La parallélisation par MPI avec 2 threads, n'a pas eu d'accélération, ce qui est un comportement attendu, puisque les cycles utilisés pour visualiser ne prennent pas beaucoup de temps, donc le code se comporte presque comme le code en séquentiel

Essai	**séquentiel**	**OpenMP**	**MPI 2 Threads**	**MPI 12 Threads**
1	62.1293	18.6515	54.6924	14.2844
2	61.1265	19.3207	58.1071	13.9446
3	59.0603	21.5172	54.2761	14.6368
4	56.065	20.5127	54.7764	14.2001
Speed up	1	2,733	1,0365	3,948

Table 1: Tableau comparatif des temps pour atteindre au moins 100 repas en séquentiel et avec OpenMP