Projet IN203

Simulation d'une colonisation d'une galaxie

Rafael Verissimo Martins

15 janvier 2021

Introduction

On se propose de paralléliser un programme qui simule la colonisation d'une galaxie par diverses civilisations technologiques à l'aide d'un modèle très simplifié et permet de voir la dynamique de colonisation de la galaxie et pourquoi le paradoxe de Fermi peut ne pas être vraiment un paradoxe…

L'objectif final du travail est d'obtenir un code parallélisé en mémoire partagée ET en mémoire distribuée. On va calculer les accélérations obtenues par ces méthodes. Dans un premier temps, on calcule le temps passé avec le code séquentiel. Ensuite, on va analyser le gain en utilisant seulement la mémoire distribuée, puis avec seulement la mémoire partagée. Enfin, on utilisera les deux méthodes ensemble pour vérifier le gain de temps.

Première partie

La première partie fait référence au code de dossier part_1. C'est le code en séquence et ici on voit les résultats obtenus dans CPU(ms)

Calcul	Affichage	Temps total
28.709 ms	2.069 ms	30.778 ms

Deuxième partie

Au départ, on n'a changé à ce stade que le programme de colonisation.cpp. Le processus zéro est alors responsable de l'affichage de l'image, et le processus 1 est responsable du calcul de la matrice. Comme la mémoire est distribuée, les deux processus doivent échanger des messages. Pour cela, on utilise la méthode d'envoi bloquant. C'est-à-dire qu'après avoir envoyé la matrice au processus 0, le processus 1 ne recalcule la matrice qu'après d'affichage. Cela correspond aux documents de la part_2.

Troisième partie

Si on veut accélérer encore plus le processus, on peut diviser encore plus le calcul de la matrice, alors que le processus zéro est responsable de l'affichage de l'image. Cependant, comme on travaille sur la mémoire distribuée, cette méthode pose un problème de mémoire. Dans les régions à la limite entre les divisions de processus, les processus peuvent être en désaccord sur certains pixels. Pour corriger cela, il faut créer 2 lignes supplémentaires sur chaque track, de sorte qu'à la fin de tous les calculs, on doit ajuster les valeurs des pixels de ces lignes. Pour cela, la communication entre les lignes est nécessaire. A la fin, lorsque tous les processus calculent leurs traces respectives et que les lignes supplémentaires sont corrigées, en utilisant MPI_GATHER, on peut réunir les informations dans le processus 0 responsable de l'affichage de l'image. Cela correspond aux documents de la part_3.

Nbp	Temps	Speedup
1	11.301 ms	1
2	10.431 ms	1.08
3	5.83 ms	1.93
4	4.09 ms	2.76
8	4.88 ms	2.31

16	3.43 ms	3.29

Quatrième partie

Grâce à des méthodes de parallélisation en mémoire partagée, on peut maintenant mettre en parallèle les calculs de la situation actuelle de chaque pixel. C'est-à-dire que l'on peut utiliser différents processus pour calculer les conditions d'une civilisation et des planètes. Il faut faire attention au fait que comme une planète peut faire des changements sur les planètes qui l'entourent, on a une condition de data race, mais particulièrement dans cette situation, ce n'est pas un problème, car le code séquentiel a le même comportement. Cela correspond aux documents de la part_4.

Threads	Calcul	Affichage	Temps total	Speedup
1	8.97 ms	2.30 ms	11.301 ms	1
2	5.66 ms	2.06 ms	7.72 ms	1.46
4	2.37 ms	2.68 ms	5.05 ms	2.23
8	5.52 ms	3.71 ms	9.254 ms	1.22
16	2.77 ms	3.96 ms	6.73 ms	1.67

Cinquième partie

En marchant pour rapprocher les deux méthodes de parallélisation, on va joindre ce qu'on a fait dans la deuxième étape avec l'étape précédente. C'est-à-dire que le thread 0 sera responsable de l'affichage et tous les autres seroa responsables des calculs. Cela correspond aux documents de la part_5.

Threads	Temps total	Speedup
1	11.301 ms	1
2	8.41 ms	1.34
4	2.94 ms	3.84
8	4.22 ms	2.67
16	1.71 ms	6.62

Finalement - **Parallélisation en mémoire partagée et distribuée**

On va utiliser un code hybride avec OpenMP et MPI, afin de réunir toutes les opérations parallèles proposées jusqu'à présent. Le code se trouve dans le dossier Version finale et les résultats sont indiqués ci-dessous.

Threads	Temps total	Speedup
1	11.301 ms	1
2x2	6.63 ms	1.70
2x4	3.43 ms	3.29
4X2	4.46 ms	2.53
4X4	42.76 ms	X
4x8	104.28 ms	X
8X4	112.14 ms	X

8X8	202.72 ms	X