OS202 - Projet Fourmi 2024

JIN Pin
LUO Yijie

8 Mars 2024

1 Q1 : Quelles sont les parties du code parallélisable lorsqu'on partitionne uniquement les fourmis ?

Lorsque nous partitionnons uniquement les fourmis pour paralléliser le code, les parties qui peuvent être parallélisées incluent principalement le traitement du comportement des fourmis et la mise à jour locale des phéromones.

1. Traitement du comportement des fourmis

Le comportement de chaque fourmi (explorer l'environnement, chercher de la nourriture, retourner au nid) peut être effectué indépendamment, sans nécessiter d'informations immédiates sur les autres fourmis. Par conséquent, nous pouvons diviser la colonie de fourmis en plusieurs groupes, chaque processus traitant le comportement d'un groupe de fourmis. Cela inclut :

— Calculer la direction du mouvement des fourmis.

— Mettre à jour la position et l'état des fourmis (si elles portent de la nourriture ou non).

— Déterminer si les fourmis ont trouvé de la nourriture ou sont retournées au nid, et mettre à jour le compteur de nourriture en conséquence.

2. Mise à jour locale des phéromones

Bien que les fourmis laissent des traces sur la carte globale des phéromones, chaque fourmi ne met à jour les phéromones qu'à sa position actuelle. Cela signifie que les fourmis dans différents processus peuvent mettre à jour les phéromones à leurs positions respectives simultanément, ce processus peut être parallélisé.

Cependant, il est important de noter que bien que chaque processus puisse calculer indépendamment les fourmis dont il est responsable et les mises à jour locales des phéromones, tous les processus doivent finalement synchroniser la mise à jour de la carte des phéromones pour s'assurer que chaque fourmi utilise une carte complète et cohérente des phéromones à l'étape suivante.

2 Q2 : Quels gains on a obtenu (speed-up) ?

Nous avons choisi de tester le code de sérialisation et le code de parallélisation dans les conditions d'une taille de labyrinthe de 25*25 et d'un total de 3000 cycles (les autres paramètres sont les mêmes que les valeurs par défaut dans le fichier ants.py). Nous calculons séparément la durée complète de chaque programme dans chaque cas. Voici les résultats des tests.

Table 1 – Résultats Expérimentaux

Nombre de processus	2	3	4	5	6	7	8
temps(s)	28.36	22.01	19.32	19.01	18.85	18.69	19.02

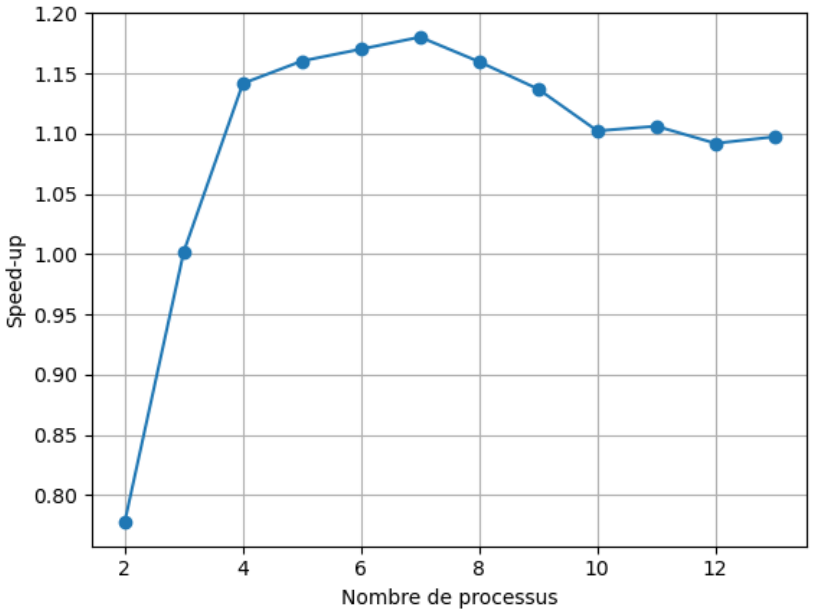
Table 2 – Résultats Expérimentaux - suivant

Nombre de processus	9	10	11	12	13	sérial.
temps(s)	19.40	20.01	19.94	20.2	20.1	22.06

On calcule donc l'accélération en fonction du nombre de processus et le trace dans la figure.

Table 3 – Résultats Expérimentaux

Nombre de processus	2	3	4	5	6	7
speed-up	0.778	1.01	1.14	1.16	1.17	1.18
Nombre de processus	8	9	10	11	12	13
speed-up	1.16	1.14	1.10	1.11	1.09	1.10



Nous avons constaté que le speed-up augmente rapidement avec le nombre de processus puis diminue lentement, atteignant son maximum avec 7 processus. Lors de l'utilisation de 2 processus (un pour l'affichage et un pour le calcul), le speed-up est inférieur à 1.

Nous pensons que lorsque le nombre de processus utilisés est faible, en plus du calcul propre au programme, le temps nécessaire pour la communication

entre les processus entraîne une durée supérieure à celle du code exécuté séquentiellement. Cependant, lorsque le nombre de processus est plus élevé, le temps consommé pour la communication entre les processus dépasse le temps économisé par l'ajout de processus, d'où la diminution progressive du speed-up.

3 Q3 : Décrire vos réflexions de comment vous voyez la mise en oeuvre du code en parallèle si on partionne en plus le labyrinthe entre les divers procesus.

Paralléliser à la fois les fourmis et les labyrinthes est un problème difficile et augmentera considérablement la complexité de la gestion des données et de la communication entre les processus. Voici Voici quelques nos réflexions sur la situation.

1. Partitionnement du labyrinthe

La première étape consiste à diviser le labyrinthe en plusieurs sections, chacune gérée par un processus différent. Cela peut être analysé grâce à la connaissance de la décomposition de domaine que nous avons apprise en classe.

2. Gestion des frontières

Parce que le labyrinthe est partitionné, chaque processus ne possède qu'une connaissance partielle de l'ensemble du labyrinthe. Les fourmis situées près des frontières entre les partitions peuvent nécessiter des informations provenant de partitions adjacentes pour décider de leur mouvement. Cela implique une communication entre les processus pour échanger des informations sur les frontières.

3. Mise à jour locale des phéromones

La mise à jour des phéromones devient plus complexe car chaque processus doit gérer la mise à jour des phéromones uniquement dans sa partition. Si une fourmi traverse la frontière vers une autre partition, elle doit communiquer avec le processus gérant cette nouvelle partition pour mettre à jour les phéromones.

4. Synchronisation des phéromones globales et des fourmis

Puisque les phéromones sont affectées par les fourmis et que les actions des fourmis sont également liées aux phéromones à leur emplacement, les processus doivent être synchronisés à certains points, notamment pour la mise à jour globale des phéromones et pour assurer la cohérence du labyrinthe et des états des fourmis à travers les processus.

4 Autres réflexions et discussions liées à ce projet

Dans le cadre de l'implémentation de l'algorithme d'optimisation par colonie de fourmis (ACO), une approche parallèle de type maître-esclave a été envisagée pour optimiser le processus de recherche. Cette stratégie reconnaît le rôle unique du processus maître (processus 0), qui diffère des autres en termes de participation au calcul. Le défi principal rencontré lors de l'adaptation de l'algorithme

ACO à un environnement parallèle réside dans la détermination efficace des mécanismes d'échange d'informations entre les processus. Plusieurs méthodes ont été considérées pour maintenir la cohérence et synchroniser la progression de la recherche de nourriture à travers les différents processus esclaves.

La première méthode se concentre sur l'âge des fourmis, garantissant leur cohérence à travers les différents processus. La seconde approche synchronise les avancées des différentes fourmis dans la recherche de nourriture, permettant au processus maître de gérer collectivement la mise à jour des phéromones une fois qu'une fourmi dans n'importe quel processus esclave trouve de la nourriture.

La sélection entre ces méthodes implique une évaluation des frais généraux associés à l'échange d'informations entre les processus, ainsi qu'à l'exécution de l'algorithme lui-même. Dans le cas général, une analyse minutieuse et un jugement éclairé du concepteur de l'algorithme sont cruciaux pour déterminer l'approche la plus efficiente, qui optimise la parallélisation et accélère le processus global de manière significative.

Dans le cadre de ce projet, nous avons opté pour la mise en œuvre et l'expérimentation de la seconde méthode. Cette décision était fondée sur une évaluation préliminaire des implications de chaque approche en termes de complexité et de surcharge de communication. Il est important de noter que, bien que nos observations et nos résultats soient prometteurs, une analyse comparative directe avec la première méthode n'a pas été réalisée. Par conséquent, nous ne formulons pas de conclusions définitives concernant la supériorité d'une approche par rapport à l'autre en termes de surcharge opérationnelle. Cependant, nos expériences indiquent que la méthode choisie présente des avantages significatifs en termes d'efficacité de la parallélisation et de la vitesse d'exécution, soulignant le potentiel de cette stratégie pour améliorer les performances de l'algorithme ACO dans des environnements de calcul parallèle.