Recherche du chemin optimal avec les algorithmes coopératifs

CARVALHO FRULANE DE SOUZA Daniel
LACERDA BENEVIDES Filipe

Palaiseau, le 08 mars 2024

1 Code

Le code avec parallélisation se trouve dans ce répertoire.
Nous avons refactorisé quelques parts du code originel pour faciliter la lisibilité du code, l'interprétation de l'image et la performance de base.

2 Discussion

2.1 Parallélisation des fourmis

Le code sera parallélisable pour toute instance de opération répété plusieurs fois avec une partie des données sans interférence avec les autres processus. À savoir, les parts pertinents du code dans ces contraintes appliqués aux fourmis seront:

1. La division de tâches des fourmis entre plusieurs objets de classe "Colony";
2. La distribution des positions des phénomènes entre les tâches des fourmis;
3. La distribution du labyrinthe entre les tâches des fourmis;
4. La division du code en une architecture qui divise les responsabilités entre le traitement des fourmis et l'affichage de l'image.

2.2 Speedup

Le speedup est une mesure de performance du code parallélisé pour rapport au code non-parallélisé, trouvée pour la formule suivante:

$$S(n) = \frac{T_s}{T(n)}$$

Où T_s est le temps original d'exécution des calculs non-parallélisés, n est le nombre de processeurs du code parallélisé, et $T(n)$ est le temps d'exécution des calculs parallélisés pour n processeurs.
Notre objectif pour le code des fourmis sera d'optimiser le temps de calcul de l'affichage du display du code séquentiel par rapport au code parallèle. Notre méthodologie de benchmark consiste en évaluer les performances dans un environnent composée par un labyrinthe de dimensions 40x40, trouvant le temps moyen écoulé entre l'affichage de images consécutifs dans le display en 3 exécutions du

code par chaque valeur de n, après la stabilisation de la route des fourmis.
Nous avons trouvé les valeurs suivantes pour différents nombres de processeurs, soit $T_s = 14.76$ ms :

Table 1: Speedup

n	$T(n)$	$S(n)$
2	12.37 ms	1.1932
3	11.77 ms	1.2541
4	12.92 ms	1.1423
8	15.15 ms	0.9743

La réduction de performance par rapport à l'augmentation de la quantité des processeurs pour n > 3 peut être expliqué pour la concurrence de mémoire entre toutes les tâches et la synchronisation des tâches dans le processeur de rank 0 occasionnées pour la partition des ressources du programme en plusieurs processeurs.

2.3 Parallélisation du labyrinthe

La parallélisation du labyrinthe pourra réduire la fréquence de création de tâches individuels en augmentant la charge de travail attendue pour chaque tâche. Pour le faire, nous pouvons diviser le labyrinthe en petites portions (jusqu'à une portion d'une "case" unitaire, seulement), responsable pour décrire le comportement de toutes les fourmis dedans.
Pour cette architecture de parallélisation, il faut aussi donner un traitement spécial en chaque bord des portions pour les fourmis qui sortent d'une portion vers une autre portion, pour éviter la corruption de valeurs utilisées entre différents processeurs. Dans ce cas, nous faisons les calculs normalement pour toutes les fourmis qui ne sortent pas d'une portion P, et pour les fourmis qui sortent d'une portion P_1 vers une portion P_2 du labyrinthe, nous mettons à jour leur nouveaux informations en P_1 et P_2 après le calcul des fourmis normales en P_1 et P_2 comme en P.
Pour bien distribuer la quantité de calculs pour chaque processeur, nous pouvons créer une liste avec tous les portions du labyrinthe contenant au moins une fourmis, trier cette liste, et donner une portion du labyrinthe à chaque processus, commençant pour la valeur plus grand. De cette manière, nous évitons qu'un processeur prendra une grande tache au fin de la liste, au même temps que les autres processeurs sont en attente.

Annexe

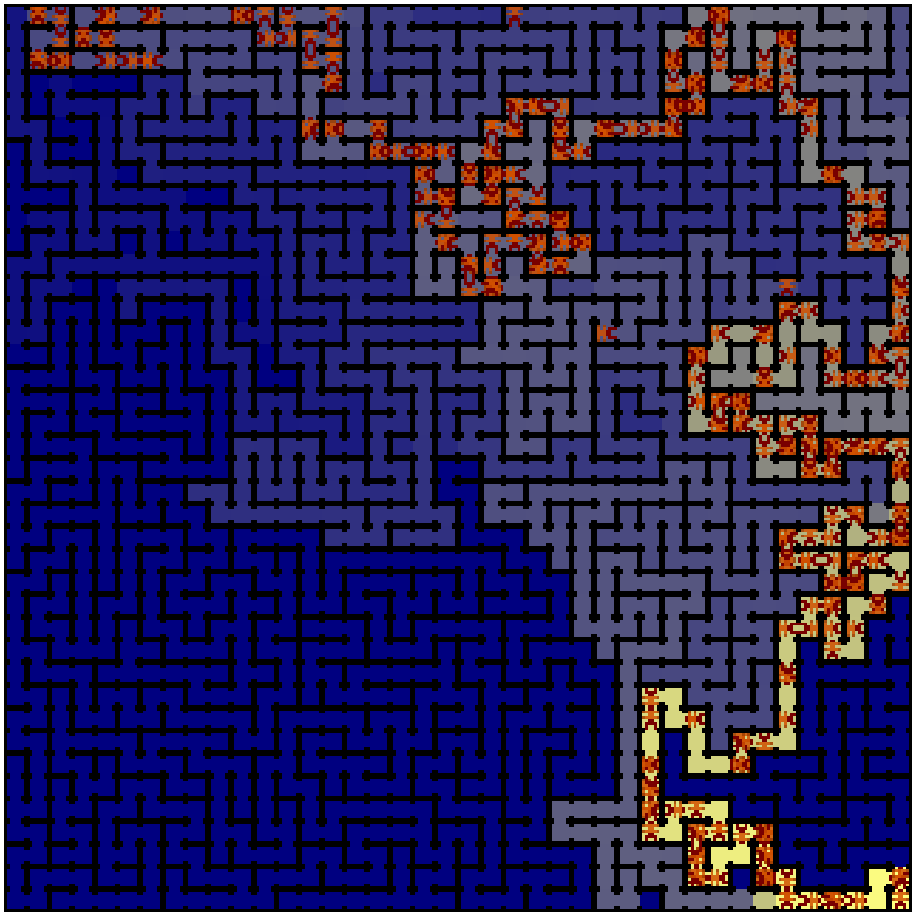

Figure 1: Chemin trouvée pour une exécution quelconque