pco25_lab06_saber_gorgerat

Auteurs : Jonathan Saber, Julien Gorgerat

Introduction au problème :

Dans ce laboratoire, nous avons dû optimiser la multiplication entre deux matrices carrées grâce à plusieurs threads. Cette approche est intéressante car elle permet d'accélérer la vitesse de résolution en calculant plusieurs valeurs en parallèle. Cependant, cela nécessite une synchronisation apportée par un moniteur de Hoare. Ainsi, nous pouvons créer une liste de tâches que les threads se partagent en assurant que chacun en exécute une différente.

Choix d'implémentation :

Pour ce laboratoire, nous avons suivi un modèle par délégation: un thread principal crée une liste de jobs qui seront effectués par les threads secondaires. Dans notre cas, nous séparons la matrice en plusieurs blocs de taille donnée, chacun de ces blocs contient une matrice de cette taille. Cette méthode nous permet d'isoler les valeurs nécessaires au calcul d'une valeur finale de la multiplication dans des blocs et donc de créer un job par valeur cherchée, qui prend uniquement les blocs nécessaires au calcul.

Chaque job contient une instance de la classe ComputeParameters, qui elle-même contient toutes les informations nécessaires aux calculs souhaités. Ces jobs sont créés au départ de notre programme puis mis dans une liste.

Afin d'assurer la synchronisation, il nous est demandé d'utiliser un moniteur de Hoare implémenté dans la classe Buffer. Nous avons donc créé une variable de condition hasJob sur laquelle les threads vont attendre. Lorsqu'un signal est émis sur cette condition, un thread en attente va alors prendre un job dans la liste des jobs créés et l'effectuer avant de se remettre en attente, et ce tant que la liste n'est pas vide.

Une deuxième condition, noJob, est utilisée par le thread principal pour attendre la fin des threads workers. Cette variable de condition reçoit un signal lorsque le dernier job de la file est terminé. Il est pertinent de noter que, malgré le fait qu'il s'agisse d'un moniteur de Hoare et non de Mesa, nous avons utilisé une boucle while, car il est possible qu'un signal soit envoyé alors que la condition n'est pas respectée (la file peut être vide alors que tous les jobs nécessaires au calcul n'y ont pas encore été ajoutés). Nous avons choisi cette approche plutôt qu'une attente active (boucle while vide), car elle permet de mettre le thread principal en attente et donc d'éviter que l'ordonnanceur ne lui donne la main alors qu'il ne fait rien, ce qui permet de privilégier les threads workers.

Tests effectués :

Nous avons effectué les tests de base fournis avec le laboratoire, que nous avons passés avec succès. Nous n'avons en revanche pas testé l'interblocage.