Quicksort multithreadé

Auteurs: Sara Camassa, Nicolas Carbonara

Introduction au problème

Ce laboratoire consiste en l'implémentation d'un algorithme de tri de type Quicksort selon un pseudocode donné, ainsi que de la partition de ce tri en plusieurs tâches accomplies par des threads, et l'exploitation d'un moniteur de Mesa pour gérer l'attente des traitements.

Choix d'implémentation

Classe Quicksort

Pour implémenter l'algorithme quicksort multi-threadé, nous utilisons une classe Quicksort qui implémente la classe abstraite générique MultithreadedSort.
Dans son constructeur, elle crée autant de thread qu'indiqué dans l'attribut `nbThreads`, avec comme routine la fonction `worker()`.

Les attributs notables spécifiques à cette classe sont:

- un vecteur de threads `threads` qui contiendra les threads créés dans le constructeur
- un pointeur `arrayToSort` sur un vecteur, qui pointera sur le tableau à trier
- une queue de fonctions `tasks`, qui contiendra des tâches à effectuer

Fonction sort

Cette fonction récupère le tableau à trier, et place dans la queue `tasks` le premier appel à la fonction quicksort, avec le tableau entier.

Elle va ensuite simplement attendre que tous les threads aient terminé, et que le tri soit donc complété.

Si le programme est lancé avec 0 threads, le premier appel à quicksort ne sera jamais exécuté, et le tri ne se fera pas.

Fonction quicksort

Cette fonction contient le tri récursif en lui-même.

Elle place d'abord le pivot à l'aide de la fonction `partition`, et sépare ensuite le tableau en deux parties à gauche et à droite du pivot. Elle s'ajoute ensuite récursivement dans la queue de tâches, une fois pour la partie gauche et une pour la droite. La récursion s'arrête lorsqu'il n'y a plus assez d'élément pour faire la séparation.
Cela permet de trier un tableau partie par partie, jusqu'à ce que tous les éléments soient triés.

Si le nombre de thread actif max est atteint, nous effectuons un quicksort sur la partie gauche et/ou droite directement (sans mettre en queue) afin de garantir que le tri ne reste pas bloqué dû à une tâche jamais exécutée (car attend sur l'exécution de la tâche inférieure).

Nous avons choisi de ne pas mettre la fonction partition dans la queue, pour garantir d'avoir un accès immédiat au pivot pour les appels au quicksort gauche et droit.

Nous protégeons la partition avec un mutex pour éviter les problèmes de concurrence sur le vecteur référencé.

fonction worker

Cette fonction permet à un thread d'attendre qu'un quicksort avec une nouvelle partition du tableau soit disponible dans la queue `tasks`, de l'exécuter, puis de continuer ainsi jusqu'à la fin complète du tri.

fonction finishtask

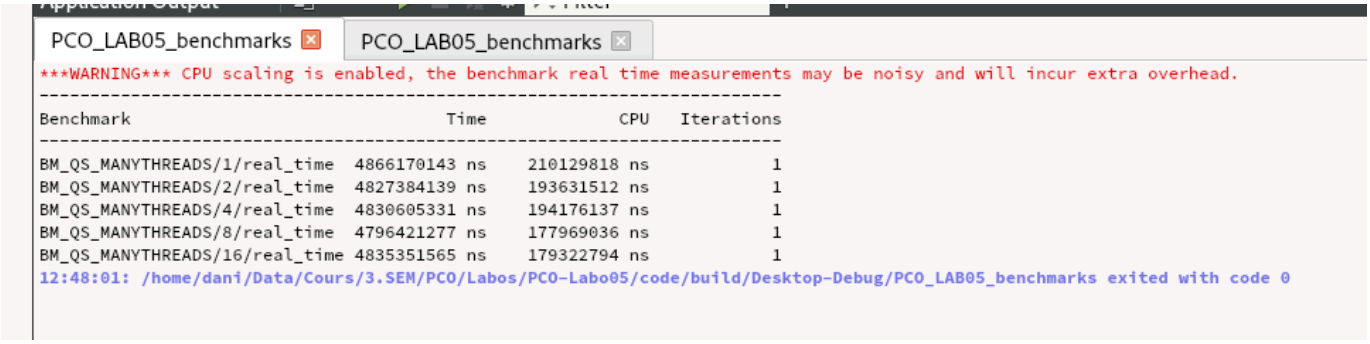
Cette fonction indique au moniteur de Mesa qu'une tâche a été terminée. Elle réveille un thread si la tâche est finie, et tous les threads si le tri a été completé.

Tests effectués

Tests

Nom	Input	Résultat
TestMoreThreadsThanSize	size = 10 nbThreads = 15	PASSED
TestSize0	size = 0 nbThreads = 10	PASSED
Test0Thread	size = 100 nbThreads = 0	PASSED
Test1Thread	size = 100 nbThreads = 1	PASSED

Benchmark


PCO_LAB05_benchmarks
PCO_LAB05_benchmarks
WARNING CPU scaling is enabled, the benchmark real time measurements may be noisy and will incur extra overhead.
Benchmark Time CPU Iterations
BM_QS_MANYTHREADS/1/real_time 4866170143 ns 210129818 ns 1
BM_QS_MANYTHREADS/2/real_time 4827384139 ns 193631512 ns 1
BM_QS_MANYTHREADS/4/real_time 4830605331 ns 194176137 ns 1
BM_QS_MANYTHREADS/8/real_time 4796421277 ns 177969036 ns 1
BM_QS_MANYTHREADS/16/real_time 4835351565 ns 179322794 ns 1
12:48:01: /home/dani/Data/Cours/3.SEM/PCO/Labos/PCO-Labo05/code/build/Desktop-Debug/PCO_LAB05_benchmarks exited with code 0