HE VD HAUTE ÉCOLE D'INGÉNIERIE ET DE GESTION DU CANTON DE VAUD

Chargés de cours : Florian Vaussard, Rick Wertenbroek

Laboratoire de Programmation Concurrente

semestre automne 2022

Prise en main des Pcothreads

Temps à disposition : 4 périodes (deux séances de laboratoire)

1 Objectifs pédagogiques

— Se familiariser avec l'environnement de développement QtCreator ;

— Réaliser un programme concurrent en C++ avec la bibliothèque pco-synchro ;

— Accélérer un calcul mathématique.

2 Cahier des charges

Nous désirons implémenter une fonction qui nous permet de déterminer si un nombre passé en paramètre est premier ou non. Bien entendu, nous sommes intéressés par une implémentation multi-threadée.

Partie 1

La première partie de cet exercice est de développer une version séquentielle de l'algorithme.

Pour ce faire, implémentez la fonction `isPrime()` de la classe `PrimeNumberDetector` qui prend en argument le nombre à tester et returne `true` si le nombre est premier et `false` dans le cas contraire.

Pour vérifier qu'un nombre entier n est premier ou non, il faut vérifier s'il peut être divisé par un des nombres compris entre 2 et $\sqrt{n}$ [1], si le nombre est divisible il n'est pas premier.

A noter que le nombre premier peut être potentiellement grand, ici un entier non signé sur 64-bits est utilisé (`uint64_t`) et peut contenir au maximum la valeur $18'446'744'073'709'551'615$ ($2^{64} - 1$)

Partie 2

Dans cette deuxième partie nous voulons améliorer les performances de notre implémentation en développant une version multi-threadée de l'algorithme. La classe `PrimeNumberDetectorMultithread` est utilisée pour ceci. Son constructeur prends le nombre de threads à utiliser et sa fonction `isPrime()` réalise l'algorithme multi-threadé.

A noter que dans votre implémentation aucun des threads ne devra utiliser la fonction exit pour terminer le programme. Pensez-donc à utiliser une politique de terminaison afin d'optimiser l'utilisation du processeur (pas de calcul pour rien, si un thread trouve un diviseur, tous les threads doivent s'arrêter. Il n'est pas nécessaire de tester si un thread a fini à chaque itération (nombre testé) mais par exemple tous les N nombres. Le choix de N est un compromis entre le temps qu'il faudra pour les threads à s'arrêter si un thread trouve un diviseur et l'impact de ce test sur l'algorithme général).

1. `https://en.wikipedia.org/wiki/Prime_number`

Partie 3

Comparez la vitesse d'exécution entre vos deux implémentations, notamment en changeant le nombre de threads. Le fichier `main.cpp` du projet main permet de lancer les deux versions avec un nombre à tester et un nombre de threads choisi grâce à deux variables.

Informations

— Les fonctions à implémenter se trouvent dans le fichier `PrimeNumberDetector.cpp` et sont définies dans le fichier `PrimeNumberDetector.h`. Vous pouvez modifier ces deux fichier pour réaliser vos implémentations.

— Trois projets Qt sont fourni pour ce laboratoire (ouvrez le projet global `PCO_Labo_2_test.pro` avec QtCreator).

1. **main** : Contient le fichier `main.cpp` où vous pouvez effectuer vos tests pour différents nombres et différents nombres de threads. Il peut aussi vous servir pour le débug.
2. **main_bench** : Ceci est un projet qui lance plusieurs benchmarks [2] et retourne les temps d'exécutions moyens sur un grand nombre d'itérations.
3. **main_test** : Ceci est un projet qui lance quelques tests unitaires [3] afin de vérifier le fonctionnement de la fonction `isPrime()`.

Questions

1. Mesurez le gain de temps produit par votre version multi-threadée en faisant varier le nombre de threads. Que remarquez-vous ?
2. Le gain de vitesse est-il linéaire avec le nombre de threads ? (Indiquez le nombre de threads alloués à la VM, il peut être changé dans les settings, ainsi que le nombre de threads de votre machine). Il y a-t-il une différence pour les nombre premiers et non premiers ?

Informations utiles

— Vous pouvez tester votre programme avec les nombres premiers suivants : 433494437, 2971215073, 68720001023, 4398050705407, 70368760954879, 18014398241046527, 99194853094755497

— La fonction `sqrt` de la libraire `cmath` renvoie la racine carrée d'un double

3 Travail à rendre

— Les modalités du rendu se trouvent dans les consignes qui vous ont été distribuées.

— La description de l'implémentation, ses différentes étapes, la manière dont vous avez vérifié son fonctionnement et toute autre information pertinente doivent figurer dans votre mini rapport.

— Inspirez-vous du barème de correction pour connaître là où il faut mettre votre effort.

— Vous pouvez travailler en équipe de deux personnes au plus.

2. `https://github.com/google/benchmark`
3. `https://github.com/google/googletest`

4 Barème de correction

Conception	25%
Exécution et fonctionnement	20%
Codage	15%
Documentation	30%
Commentaires au niveau du code	5%
Robustesse	5%