Travaux Pratique 5

Ce sujet est en lien avec le troisième chapitre du cours, et concerne donc les *patrons de conceptions parallèles*. Les mêmes commentaires que ceux des derniers TP s'appliquent ici aussi.

Dans cette séance, l'objectif est donc d'implanter le patron PARTITION sur une machine classique multicœur, grâce à la gestion des threads proposée par C++. Vous allez démarrer avec une version séquentielle, afin de vérifier que l'ordre des sous-listes est bien respecté. Ensuite, vous écrirez une version parallèle. Enfin, vous terminerez cette séance avec l'algorithme du tri base, qui repose sur l'utilisation du patron PARTITION.

Rappelez-vous qu'un algorithme parallèle écrit pour p processeurs peut être simulé sur une machine parallèle contenant seulement $p' < p$ processeurs … Souvent le temps d'exécution en est réduit, mais cela améliore l'efficacité !

Votre travail est à rendre (il le sera à chaque fois) sous la forme du code (et uniquement la partie « student ») accompagnée d'un rapport au format PDF, le tout dans une archive compressée au format ZIP. Ne respecter pas ces contraintes, et votre note en sera diminuée de quelques points.

Vous ne devez modifier que ce qui se trouve dans le répertoire « `./student/` » !

Notez que chaque exercice vient avec un squelette, qui s'occupe de la ligne de commande, du lancement de votre code (défini dans une classe particulière), et d'une vérification sommaire du résultat (lorsque c'est possible). Utilisez l'option « `-h` » ou « `--help` » pour connaître le fonctionnement de la ligne de commande …

Deux scripts sont fournis pour compiler : sous Windows, il s'agit de « `./build.bat` » qui génère les exécutables dans le répertoire « `./win32/Release` » ; pour les autres systèmes, utilisez « `./build.sh` » qui génère les exécutables dans « `./linux/` ». Ces scripts utilisent « `cmake` », et soit `GCC` sous linux/mac, soit `Visual C++ 17` sous Windows (mais la version 19 fonctionnera aussi). Leur comportement a été testé sous Windows, WSL/Debian, et Linux Mint.

NB : dans les exercices suivants, utilisez l'option « -s=16 » afin de faciliter le déverminage.

Exercice 1

Cet exercice est relativement simple : il consiste à écrire le patron PARTITION en version séquentielle. Attention à respecter l'ordre des sous-listes !

Par exemple, le résultat du patron PARTITION sur la liste de valeurs $\{1,2,3,4,5,6,7,8\}$ avec les prédicats $\{0,0,0,0,1,1,1,1\}$ est $\{5,6,7,8,1,2,3,4\}$. Notamment, la liste $\{5,6,7,8\}$ conserve son ordre …

Exercice 2

Ecrivez une première version du tri base en utilisant votre patron PARTITION en version séquentielle. Notez les performances en variant la taille de l'entrée. Pas terrible pour de petites valeurs, n'est-ce pas ?

Exercice 3

Ecrivez une seconde version du tri base en utilisant les patrons SCAN inclusif et exclusif en version parallèle que vous avez écrit lors de la séance précédente, ainsi que le patron MAP (`transform`) du TP 3. Vous avez étudié l'algorithme PRAM en cours, ainsi qu'en TD (rapidement, certes). Quel est l'impact sur les performances ? Que se passerait-il en utilisant un pool de threads dans nos patrons (tracez la courbe du temps de calcul en fonction de la taille du tableau à trier, vous comprendrez la question).

Exercice 4

Ecrivez le patron PARTITION en version parallèle sans utiliser les patrons précédents (dans l'espoir d'être plus rapide !). Le code est à écrire dans le fichier `partition.h`, en utilisant `OPP::nbThreads` threads. Quelle est l'accélération pour n éléments ?

Exercice 5

Ecrivez une troisième et dernière version du tri base en utilisant vos patrons MAP et PARTITION en versions parallèles. Quel est l'impact sur les performances ?