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

Yann Thoma
Olivier Cuisenaire
Florian Vaussard

# Laboratoire de Programmation Concurrente

semestre automne 2024

# Quicksort multithreadé

Temps à disposition : 4 périodes

## 1 Objectifs pédagogiques

— Réaliser un algorithme de tri réparti en plusieurs tâches.
— Réaliser un programme en exploitant un moniteur de Mesa.

## 2 Cahier des charges

Nous désirons réaliser un algorithme de tri d'un tableau d'entiers via l'algorithme Quicksort, dont le pseudo-code est proposé ci-dessous [1].

```
// Sorts (a portion of) an array, divides it into partitions, then sorts those
algorithm quicksort(A, lo, hi) is
  // Ensure indices are in correct order
  if lo >= hi || lo < 0 then
    return

  // Partition array and get the pivot index
  p := partition(A, lo, hi)

  // Sort the two partitions
  quicksort(A, lo, p - 1) // Left side of pivot
  quicksort(A, p + 1, hi) // Right side of pivot

// Divides array into two partitions
algorithm partition(A, lo, hi) is
  pivot := A[hi] // Choose the last element as the pivot

  // Temporary pivot index
  i := lo

  for j := lo to hi - 1 do
    // If the current element is less than or equal to the pivot
    if A[j] <= pivot then
      // Swap the current element with the element at the temporary pivot index
      swap A[i] with A[j]
      // Move the temporary pivot index forward
      i := i + 1

  // Swap the pivot with the last element
  swap A[i] with A[hi]
  return i // the pivot index
```

Comme on le voit dans ce pseudo-code, l'algorithme est récursif, avec deux appels à la fonction de base par elle-même. Il est donc possible de lancer deux threads à chaque fois et d'attendre leur

1. Tiré de `https://en.wikipedia.org/wiki/Quicksort`.

terminaison. Cette solution, bien que fonctionnellement correcte, risque fort de ne pas être réellement efficace car lancer un thread a un coût computationnel non négligeable.

L'idée est donc ici de lancer un certain nombre de threads, et que ceux-ci attendent qu'il y ait un traitement à faire, le fassent, puis indiquent qu'ils ont terminé et attendent à nouveau une nouvelle tâche. La synchronisation doit se faire grâce à un moniteur de Mesa, donc des `PcoConditionVariable`.

La classe à compléter hérite d'une classe abstraite `MultithreadedSort<T>`, et offre une unique fonction publique, en plus du constructeur. Le constructeur prend en paramètre le nombre de threads qui doivent pouvoir être lancés, et la fonction `sort()` prend en paramètre un vecteur à trier. Le tri se fait directement dans le vecteur, il n'y a pas de copie qui est faite.

```
template<typename T>
class Quicksort: public MultithreadedSort<T>
{
public:
    Quicksort(unsigned int nbThreads) :
        MultithreadedSort<T>(nbThreads) {}

    void sort(std::vector<T>& array) override {
        // TODO
    }
};
```

Il est évident que le système doit être *nettoyé* avant la fin de la fonction `sort()`, et donc que les allocations mémoire devraient être désallouées et les threads lancés terminés.

Le code fourni offre deux projets : un avec des tests automatisés, et un avec des benchmark. Vous noterez que les tests sont vides mais qu'il y a de quoi en ajouter. Nous laissons ceci à votre responsabilité. Pour les benchmarks, ils lancent un tri avec 1, 2, 4, 8 et 16 threads, et vous permettent d'observer l'efficacité de votre solution

Il est nécessaire d'installer la librairie **google-benchmark** afin de pouvoir compiler ce dernier projet. Par exemple sur Ubuntu :

```
sudo apt-get install libbenchmark-dev
```

ou sur Fedora :

```
sudo dnf install google-benchmark-devel
```

## 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 un fichier nommé `rapport.pdf`.

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

— Vous devez travailler en équipe de deux personnes.

— L'archive à rendre doit être générée avec le script de rendu `pco_rendu.sh`

## 4 Barème de correction

| | |
|---|---|
| Conception | 50% |
| Exécution et fonctionnement | 5% |
| Codage | 10% |
| Documentation et analyse | 35% |