# Tri Multithreadé

Burkhalter - Lienhard

Version 1.0
5/25/2010 11:29:00 PM

# Table des matières

Tri Multithreadé ..... 3
Introduction ..... 3
Réalisation ..... 3
Tests ..... 3
test_1 ..... 4
test_2 ..... 4
test_3 ..... 4
Bugs ..... 4
Mesures ..... 4
Comparaison ..... 4
Index des fichiers ..... 5
Documentation des fichiers ..... 6
main.c ..... 6
Description ..... 7
Index ..... 11

# Tri Multithreadé

Cette documentation décrit le programme Tri Multithreadé qui doit permettre le tri d'un tableau réparti en plusieurs tâches. Cela a été mis en place dans le laboratoire n°6 du cours PCO.

## Introduction

Le but de ce programme est la réalisation d'un programme permettant le tri d'un tableau à l'aide de plusieurs threads. Pour ce faire, le tableau doit être divisé en différentes zones, et chacune de ces zones va être triée par un thread. Entre chaque zone existe une cellule commune aux deux zones adjacentes. La principale difficulté consiste en la gestion de ces cellules communes.

Afin de réaliser le tri, l'algorithme de tri à bulles sera utilisé. De plus, les informations concernant la taille du tableau ainsi que du nombre de threads doivent être insérées par l'utilisateur.

## Réalisation

Les différentes étapes nécessaires à la réalisation de notre programme, ainsi que la manière dont celles-ci ont été implémentées vont être expliquées dans la section à venir.

Afin d'aboutir aux résultats souhaités dans ce projet, nous avons tout d'abord dû créer un tableau contenant des valeurs aléatoires. Pour ce faire nous avons déclaré un pointeur sur int, se qui nous permet de déclarer un tableau de taille variable et ainsi laisser l'utilisateur entrer la taille du tableau souhaité. La même démarche a été réalisée pour définir le nombre de threads. Une fois la taille du tableau définie, nous avons rempli les cellules de celui-ci de manière aléatoire à l'aide de la fonction rand().

Dès le tableau peuplé, la première difficulté que nous avons rencontrée à consisté à délimiter et définir la taille de zone attribuée à chaque thread. Pour ce faire, nous avons effectué les calculs suivants:

tailleZone = (TailleTableau + (nbThreads - 1)) / nbTreads

Ce premier calcul nous permet d'obtenir la taille de base de chaque zone.

nbZoneSup = (tailleTableau + (nbThreads - 1)) % nbThreads

Ce second calcul nous permet d'obtenir le nombre de zone qui vont avoir une cellule en plus que la taille de base.

Une fois le tableau rempli et les zones attribuées aux threads, nous avons attaqué la réalisation du tri à proprement parlé. Pour ce faire nous avons crée une fonction, nommé tache_tri(), et qui sera effectuée par chacun des threads afin de trier sa partie du tableau. Cette fonction permet à chaque thread de gérer sa partie du tableau. Chaque partie est triée à l'aide de l'algorithme de tri bulle. Lorsqu'un thread arrive à l'une ou l'autre de ses cellules communes à un autre thread (cellules critiques), celui-ci prend le mutex sur cette cellule avant de la traiter, afin d'assurer l'exclusion mutuelle. Lorsqu'un thread a terminé le tri de sa partie, il se met en attente et sera réveillé par un autre thread, si celui-ci a modifié la valeur se trouvant dans la cellule commune. Chaque thread travaille de la manière jusqu'à ce que le tri soit terminé, se qui sera le cas lorsque la dernière partie sera triée et donc que tous les threads seront en attentes.

Finalement pour nous assurer que le tri a correctement eu lieu, nous affichons le tableau final à l'aide de la fonction **afficherTableau()**.

## Tests

Afin de tester que le cahier des charges est respecté nous avant réalisé les tests suivants:

### test_1

Le premier test à consisté à tester que l'utilisateur ne peux pas introduire un nombre de thread supérieur au nombre de cellules dans le tableau.

Résultat: Il est redemandé à l'utilisateur d'effectuer sa saisie tant que les valeurs ne sont pas cohérentes, se qui correspond à nos désires.

### test_2

Le deuxième test à consisté à contrôler que la séparation du tableau en différentes zones se fasse de manière correcte. Pour tester cela, nous avons affiché notre tableau d'indices et avons vérifié que la taille des zones soit correcte.

Résultat: La taille des zones est correcte, à savoir une variance du nombre de cellule de un au maximum.

### test_3

Ce test à consisté à tester que le tri du tableau se fasse de manière correcte et cela avec des tailles de tableau différents ainsi qu'avec des nombres de thread petit ou grand. Pour tester cela, nous avons testé le tri de tableaux de plus en plus grand et avons utilisé des nombres de threads variables.

Résultat: Le tri du tableau se fait de manière correcte lors du tri de tableaux de petites tailles à taille moyenne, cependant lors du tri de tableaux de grandes tailles ou ayant un nombre important de threads, le tri ne s'effectue pas et le programme reste bloqué. (Voir section Bugs)

## Bugs

Notre programme contient toujours un objectif qui ne fonctionne pas parfaitement, dans le fait qu'il arrive lors de tri de tableaux de grande taille ou ayant un nombre important de threads, que notre programme ne termine pas le tri des cellules et se bloque en cours d'exécution.

## Mesures

Comme demandé pour le laboratoire, nous avons mis en place un chronomètre permettant de mesurer le temps nécessaire pour trier un tableau à l'aide de l'algorithme multithreads que nous avons mis en place ainsi qu'à l'aide d'un simple tri bulle.

### Comparaison

La comparaison des différents temps nécessaires pour le tri de tableaux nous montre clairement que le tri multithreadé que nous avons mis en place nécessite plus de temps que la variante du simple tri bulle. Cela doit venir du fait que notre prise de temps pour le tri multithreadé englobe et comprend la création des threads, se qui nécessite passablement de temps. De plus l'attente des différents threads sur les mutex coûte aussi du temps.

# Index des fichiers

## Liste des fichiers

Liste de tous les fichiers documentés avec une brève description :

**main.c** ..... 6

# Documentation des fichiers

## Référence du fichier main.c

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <time.h>
#include <stdbool.h>
```

### Fonctions

- void **saisieClavier** ()
- void **remplirTableau** (int *tab, int length)
- void **afficherTableau** (int *tab, int length)
- void **BubbleSort** (int *tab, int length)
- bool **test_BubbleSort** (int *tab, int length)
- void **remplir_tableau_indices** (int *tab, int length)
- void **permuter** (int *element_1, int *element_2)
- void * **Tache_tri** (void *arg)
- void **allocationMemoire** ()
- void **initialisationMutex** ()
- void **liberationMemoire** ()
- void **creationThreads** ()
- void **attenteFinThreads** ()
- int * **copierTableau** (int *tableau, int **TAILLE_TABLEAU**)
- int **main** (void)

### Variables

- int * **tableau1**
- int * **tableau2**
- int * **tableau_indices**
- pthread_t * **tableau_threads**
- int **NB_THREADS**
- int **TAILLE_TABLEAU**
- bool **saisieOK** = false
- bool * **threadTrie**
- bool * **estEnAttente**
- pthread_mutex_t * **mutex**
- pthread_mutex_t * **threadAttente**
- pthread_mutex_t **mutex_attente**
- int **nbAttente** = 0
- clock_t **debut_tri_threads**
- clock_t **fin_tri_threads**
- clock_t **debut_tri_normal**
- clock_t **fin_tri_normal**
- double **temps_threads**
- double **temps_sans_thread**

## Description détaillée

**Auteur:**
Steve Lienhard et Arnaud Burkhalter

**Date:**
25.04.2010

**Version:**
1.0

# Description

Ce fichier met en place un programme de tri multithreadé basé sur l'algorithme de tri Bulles.

---

## Documentation des fonctions

**void afficherTableau (int * *tab*, int *length*)**

But : Le but de cette fonction est d'afficher le contenu d'un tableau passé en paramètre.

Paramètre(s): tab : pointeur sur int passé à la fonction.

Ce pointeur fait référence au tableau de valeurs entières qu'il faut afficher.
length: Valeur de type entière indiquant la dimension du tableau.

**void allocationMemoire ()**

But : Le but de cette fonction consiste à allouer l'espace mémoire nécessaire aux variables dynamiques utilisées dans notre programme.

**void attenteFinThreads ()**

But : Cette fonction effectue un join sur chaque thread afin que le programme ne se termine pas avant que tous les threads aient fini leur tâche.

**void BubbleSort (int * *tab*, int *length*)**

But : cette fonction effectue le tri du tableau selon l'algorithme de tri bulle.

Paramètre(s): tab : pointeur sur int passé à la fonction.

Ce pointeur fait référence au tableau de valeurs entières qu'il trier.

length :Valeur de type entière indiquant la dimension du tableau.

**int* copierTableau (int * *tableau*, int *TAILLE_TABLEAU*)**

But : Le but de cette fonction est de retourner une copie du tableau passé en paramètre.

Paramètre(s): tableau : pointeur sur int représentant le tableau que l'on souhaite copier.

taille_tab : dimension du tableau passé en paramètre.

**void creationThreads ()**

But : Cette fonction permet de créer un nombre de threads équivalent au nombre de threads souhaité par l'utilisateur.

**void initialisationMutex ()**

But : Comme son nom l'indique, cette fonction à pour objectif d'initialiser les mutex utilisés par les différents threads afin d'assurer l'exclusion mutuel entre eux.

**void liberationMemoire ()**

But : Cette fonction a pour but de restituer l'espace mémoire qui a précédemment été alloué à l'aide de malloc.

**int main (void)**

But : Fonction principale permettant l'exécution du programme. Celle-ci fait appel aux autres fonctions précédemment implémentées. Une valeur entière sera retournée, représentant si la fonction s'est terminée de manière correct ou non.

**void permuter (int * *element_1*, int * *element_2*)**

But : Cette fonction permet de permuter les valeurs de deux cellules passées en paramètre.

Paramètre(s): element_1 : élément de type int dont on souhaite affecter la valeur à element_2.

element_2 : élément de type int dont on souhaite affecter la valeur à element_1.

**void remplir_tableau_indices (int * *tab*, int *length*)**

But : Le but de cette fonction est de définir la zone d'action de chaque thread, en d'autre termes d'attribuer un certain nombre de cellules à chaque thread prenant part au tri du tableau. Ces valeurs d'indices sont stockées dans la variable tableau_indices.

Paramètre(s): tab : pointeur sur int passé à la fonction.

Ce pointeur fait référence au tableau de valeur entières qu'il va falloir trier.

length: Valeur de type entière indiquant la dimension du tableau.

**void remplirTableau (int * *tab*, int *length*)**

But : Cette fonction a pour but de peupler un tableau de valeur aléatoire. Les valeurs sont générées aléatoirement dans un intervalle de 0 à 100.

Paramètre(s): tab : pointeur sur int passé à la fonction.

Ce pointeur fait référence au tableau de valeurs entières qu'il faut remplir aléatoirement.

length: Valeur de type entière indiquant la dimension du tableau.

**void saisieClavier ()**

But : Le but de cette fonction est d'effectuer la saisie de valeurs au clavier par l'utilisateur. Il est demandé à l'utilisateur d'introduire le nombre de cellules que va devoir contenir le tableau, ainsi que du nombre de threads nécessaire pour traiter celui-ci.

**void* Tache_tri (void * *arg*)**

But : Fonction qui sera exécutée par un thread traitant le tri d'une certaine zone du tableau.

Paramètre(s): arg : pointeur passé à la fonction. Il est utilisé dans notre fonction pour passer le numéro du thread. Il nous est donc utile pour associer un thread à l'une des zones du tableau.

**bool test_BubbleSort (int * *tab*, int *length*)**

But : Le but de cette fonction est de contrôler que le tri d'un tableau a correctement été effectué.

Paramètre(s): tab : pointeur sur int passé à la fonction.

Ce pointeur fait référence au tableau que l'on souhaite tester.

length :Valeur de type entière indiquant la dimension du tableau.

---

## Documentation des variables

**clock_t debut_tri_normal**

Temps processeur auquel le tri sans thread démarre

**clock_t debut_tri_threads**

Temps processeur auquel le tri avec threads démarre

**bool* estEnAttente**

Tableau indiquand si le thread i est en attente

**clock_t fin_tri_normal**

Temps processeur auquel le tri sans thread s'arrête

**clock_t fin_tri_threads**

Temps processeur auquel le tri avec threads s'arrête

**pthread_mutex_t* mutex**

Mutex garantissant l'exlusion mutuelle quand deux threads veulent accéder à une seule case

**pthread_mutex_t mutex_attente**

Mutex qui permet de modifier nbAttente

**int NB_THREADS**

Constante qui représente le nombre de threads qui vont trier le tableau

**int nbAttente = 0**

nb de threads en attente

**bool saisieOK = false**

Bool indiquant si la saisie de l'utilisateur est correcte ou non

**int* tableau1**

Tableau contenant les nombres à trier

**int* tableau2**

Copie du tableau contenant les nombres à trier

**int* tableau_indices**

Tableau qui contient les indices des différentes zones du tableau principal

**pthread_t* tableau_threads**

Tableau qui contient les threads qui trient le tableau principal

**int TAILLE_TABLEAU**

Constante qui représente la taille du tableau

**double temps_sans_thread**

Temps d'exécution du simple tri bulle

**double temps_threads**

Temp d'exécution pour le tri avec thread

**pthread_mutex_t* threadAttente**

Mutex qui permet d'attendre la fin du tri

**bool* threadTrie**

Tableau indiquant si le thread i est trié

# Index


afficherTableau
  main.c, 6
allocationMemoire
  main.c, 6
attenteFinThreads
  main.c, 6
BubbleSort
  main.c, 6
C:/Users/You/Arnaud/HEIG/2eme/PCO/Laboratoires
  /Labo6_tri/main.c, 5
copierTableau
  main.c, 6
creationThreads
  main.c, 6
debut_tri_normal
  main.c, 8
debut_tri_threads
  main.c, 8
estEnAttente
  main.c, 8
fin_tri_normal
  main.c, 8
fin_tri_threads
  main.c, 8
initialisationMutex
  main.c, 7
liberationMemoire
  main.c, 7
main
  main.c, 7
main.c
  afficherTableau, 6
  allocationMemoire, 6
  attenteFinThreads, 6
  BubbleSort, 6
  copierTableau, 6
  creationThreads, 6
  debut_tri_normal, 8
  debut_tri_threads, 8
  estEnAttente, 8
  fin_tri_normal, 8
  fin_tri_threads, 8
  initialisationMutex, 7
  liberationMemoire, 7
  main, 7
  mutex, 8
  mutex_attente, 8
  NB_THREADS, 8
  nbAttente, 8
  permuter, 7
  remplir_tableau_indices, 7
  remplirTableau, 7
  saisieClavier, 7
  saisieOK, 8
  tableau_indices, 8
  tableau_threads, 9
  tableau1, 8
  tableau2, 8
  Tache_tri, 7
  TAILLE_TABLEAU, 9
  temps_sans_thread, 9
  temps_threads, 9
  test_BubbleSort, 7
  threadAttente, 9
  threadTrie, 9
mutex
  main.c, 8
mutex_attente
  main.c, 8
NB_THREADS
  main.c, 8
nbAttente
  main.c, 8
permuter
  main.c, 7
remplir_tableau_indices
  main.c, 7
remplirTableau
  main.c, 7
saisieClavier
  main.c, 7
saisieOK
  main.c, 8
tableau_indices
  main.c, 8
tableau_threads
  main.c, 9
tableau1
  main.c, 8
tableau2
  main.c, 8
Tache_tri
  main.c, 7
TAILLE_TABLEAU
  main.c, 9
temps_sans_thread
  main.c, 9
temps_threads
  main.c, 9
test_BubbleSort
  main.c, 7
threadAttente
  main.c, 9
threadTrie

main.c, 9