

# Architecture Multicœur

UNIVERSITE VIRTUELLE DU BURKINA FASO

OUEDRAOGO Ibrahim & BAZEGA Christophe | Projet de Architecture  
Multicœur et Introduction au HPC | 25 août 2020

## TABLE DES MATIERES

Introduction.....	2
I. Présentation du projet.....	2
II. Choix du langage de programmation.....	2
III. Structuration du projet .....	3
IV. Méthode d'exécution du projet.....	4
1. Execution Sequenciel.....	4
2. Execution Posix.....	4
3. Execution avec OpenMP .....	5
V. Comparaison des résultats des exécutions .....	5
VI. Interprétation des résultats d'exécution.....	6
Conclusion .....	7

# INTRODUCTION

Afin de mettre en pratique nos acquis sur le cours d'Architecture Multicœur et Introduction au HPC. Il nous a été donné un projet qui s'articule sur les principes de programmations multithreads articulant sur un langage de la famille C.

## I. PRESENTATION DU PROJET

Il est demandé dans ce projet de :

- ✚ D'initialiser dynamiquement 02 tableaux d'entiers de  $N^1$  d'entrées respectivement T1 et T2 avec des valeurs aléatoires compris entre -10 et 10 ;
- ✚ De manière séquentiels initialiser un tableau T3 à N entrées de tel sorte que chaque indice i de ce tableau correspond au produit respectif T1 et T2 à l'indice i et calculer la somme et la moyenne des éléments de T3 ;
- ✚ Proposer une implémentation Posix et OpenMP d'initialisation et de calcul de T3 tout en calculant la somme et moyenne des entrées de T3.

Pour la mise en œuvre du projet nous devons faire un choix entre le C et C++.

## II. CHOIX DU LANGAGE DE PROGRAMMATION

Choisir un langage de programmation entre le C et C++ revient à choisir le paradigme de programmation adéquat pour ce projet. La programmation orienté objet (C++) est

---

<sup>1</sup>  $N = 1\,000\,000$

un paradigme de programmation où le programme principal est divisé en petits objets en fonction du problème et les données et fonctions de chaque objet individuel agissent comme une entité. Nous avons opté pour la programmation procédurale où le programme principal est divisé en petites parties selon les fonctions et chaque fonction peut contenir des données différentes car nous jugeons qu'avec le langage C, nous sommes plus proche du système en C qu'en C++ et que nous pouvons bien structurer le projet aussi bien que le C++.

### III. STRUCTURATION DU PROJET

Nous avons adopté pour ce projet le principe de la programmation modulaire qui offre des avantages multiples tels que :

- ✚ La réduction de la taille des programmes sources améliorant la lisibilité et réduisant les temps de compilation ;
- ✚ La structuration accrue de l'application par la conception de modules fonctionnellement indépendants. Ceci permet de réaliser une encapsulation comparable par certains aspects à celle de la programmation orientée Object ;
- ✚ Le développement du code source des modules peut être attribuer à d'autres personnes différentes pour des tests ;
- ✚ La Limitation ou la suppression des doublons de code identique, ce qui facilite la maintenance du programme (principe de programmation DRY<sup>2</sup>) ;
- ✚ La facilitation de la réutilisabilité du code source comme bibliothèque dans les futurs projets si besoins y est.

---

<sup>2</sup> Don't Repeat Yourself

## IV. METHODE D'EXECUTION DU PROJET




En rappel, il est demandé pour le projet, une exécution Posix, OpenMP et séquentiel pour l'initialisation et les calculs de la somme et moyenne des indices du tableau d'entiers de N éléments de T3.

### 1. EXECUTION SEQUENCIEL

L'exécution séquentiel est représentée au niveau du code source par la fonction ***calculer\_t3*** qui prend en entrée 03 paramètres qui sont des pointeurs vers des tableaux T1, T2 et T3 avec T1 et T2 déjà initialisé. Dans le corps de la fonction, nous parcourons sur les indices afin d'initialiser le tableau T3 tout en sommant sur la valeur t3 a l'indice de la boucle. Ensuite calculons la moyenne qui est le rapport entre la somme et N.

### 2. EXECUTION POSIX

L'exécution Posix est représentée au niveau du code source par la fonction ***calculer\_t3\_posix*** qui prend en entrée 04 paramètres qui sont des pointeurs vers des tableaux T1, T2, T3 et le nombre de thread de la machine avec T1 et T2 déjà initialisé. Dans le corps de la fonction, nous :

-  Créons un tableau de thread de nombre de thread plus un ;
-  Nous repartitionnons équitablement (nous procédons à une division entière) les tâches entre les threads et s'il y'a du reste nous assignons au premier thread à se libérer ;
-  A la fin de chaque thread, nous récupérons la somme cumulée des indices des indices du T3 initialiser dans sa partition qu'on somme avec la somme globale défini préalablement ;



A la fin des toutes les threads, nous calculons la moyenne.

### 3. EXECUTION AVEC OPENMP

L'exécution OpenMP est représentée au niveau du code source par la fonction *calculer\_t3\_omp*. Une copie de la fonction calcul\_t3 a l'exception de la directive *#pragma omp parallel for reduction(+:val)*.

## V. COMPARAISON DES RESULTATS DES EXECUTIONS

Nous avons lancer les différents exécutions (séquentielle, Posix et OMP) dans un même programme pour comparer les temps d'exécutions (nous avons implicitement augmenter la valeur de N afin de mieux distinguer les écarts).

```
##### N = 10000000 #####
Votre machine a 4 Coeurs dont 8 Threads
Initialisation du tableau T1
Initialisation du tableau T2

##### Calcul sequencielle de T3 #####

Somme = 2576358
Moyenne = 0.257636
Temps d'operation 0.045000
##### Calcul POSIX de T3 #####

Somme = 2576358
Moyenne = 0.257636
Temps d'operation 0.019000
##### Calcul OpenMP de T3 #####

Somme = 2576358
Moyenne = 0.257636
Temps d'operation 0.029000

-----
Process exited after 0.7431 seconds with return value 0
Appuyez sur une touche pour continuer...
```

Figure 1: Résultat d'exécutions des taches suivant les différents types d'exécutions

La figure ci-dessus représente une exécution de notre programme. Nous pouvons les classer par ordre croissant du temps d'exécution de la fonction :

1. Posix
2. OpenMP
3. Séquentielle

En vue des résultats obtenue, l'on se pose la question à savoir en quoi les écarts entre l'exécution OpenMP et l'exécution Posix est si élevé ?

## VI. INTERPRETATION DES RESULTATS D'EXECUTION

### L'exécution séquentielle

Comme son nom l'indique, tous le programme s'exécute sur un seul thread. So avantage est que le programmeur reste focalisé uniquement sur le code métier sans avoir à se soucier de la manière dont le code sera exécuté coté processeur. Ce qui explique le temps d'exécution élevé de la fonction séquentielle.

### OpenMP

L'OpenMP est une librairie qui apporte du parallélisme a un code écrit avec un algorithme séquentiel sans toucher à la logique métier. Elle fait beaucoup de chose qu'on ne code pas et/ou ne voit pas<sup>3</sup>. Il est proposé dans la plupart des compilateurs. Il apporte donc une grande portabilité<sup>4</sup>. Il a pour inconvénient est qu'il fait beaucoup

---

<sup>3</sup> Création de Pool de thread, distribution du travail, attentes lors des section critiques et bien d'autres.

<sup>4</sup> Peu importe la plateforme, le compilateur, le processeur ou que le programme s'exécute sur un portable...

de chose qui ne sont pas nécessaire. Ce qui explique sa lenteur relative à l'exécution Posix.

### L'exécution Posix

L'exécution de la fonction Posix est la plus rapide des tous car elle est implémentée au niveau de l'algorithme, le développeur a la main mise sur la manière d'exécuter les threads. Cependant il est pratiquement impossible de rendre un algorithme séquentiel en Posix sans toucher à la logique métier. En plus il n'est pas implémenté nativement dans certains compilateurs.

## CONCLUSION

Le travail pratique nous a permit de consolider nos acquis théoriques sur le cours de l'architecture des multicœurs et introduction à OpenMP. Ces acquis changeront notre manière d'appréhender nos futurs projets.