UCLouvain

École polytechnique de Louvain

LINFO1152 - Systèmes Informatiques

Projet 1 : Programmation multi-threadée et évaluation de performances

---

Groupe 24 (vendredi)

Auteur :
Cédric Kheirallah
Noma :
0462-19-00

Enseignant :
Étienne Rivière

Assistants :
François de Keersmacker
Maxime Piraux
Tom Rousseaux
Nikita Tyunyayev

2022-2023

1 Introduction

1.1 Consignes

Dans le cadre de ce projet, il nous a été demandé d'implémenter, d'évaluer puis d'expliquer les performances de différentes applications multi-threadées.
Les applications sont celles vues au cours et au TD :

— problème des philosophes

— problème des producteurs-consommateurs

— problème des lecteurs-écrivains

L'évaluation de ces programmes se fait via leur temps d'exécution total tout en variant le nombre de threads d'exécution (1, 2, 4, 8, 16, 32 et 64)

1.2 Problème de groupe/section drama

Ce projet se faisait normalement par groupe de deux mais ayant constaté que mon coéquipier n'avait encore soumis **aucun** code fonctionnel sur le repository du groupe, malgré que j'étais malade avec 40°C de fièvre et qu'il était censé l'avoir fait 5 jours plus tôt, j'ai décidé de dissoudre le groupe après discussion afin que chaque membre du groupe reçoive un résultat à la hauteur de son travail.

1.3 Au sujet de la partie 2

Si vous lisez ce paragraphe cela signifie que je n'aurais pas eu le temps de la faire pour les raisons expliquées ci-dessus et que j'ai préféré me concentrer sur le code de la partie 1 et le rapport.
La conclusion portera donc sur les résultats de la partie 1 du projet.

2 Partie 1

2.1 Problème des philosophes

Le problème des philosophes aussi connu sous le nom du problème de l'exclusion mutuelle considère le cas de plusieurs threads qui se partagent une ressource commune qui doit être manipulée de façon exclusive.
Chaque philosophe est un thread qui peut soit "penser", soit "manger". Pour manger il lui faut deux baguettes or il y a autant de baguettes que de philosophes, ils doivent donc se partager les ressources pour "manger" chacun à tour de rôle. Lorsqu'ils n'ont pas accès aux baguettes, les philosophes "pensent".

2.1.1 Résultats et explications

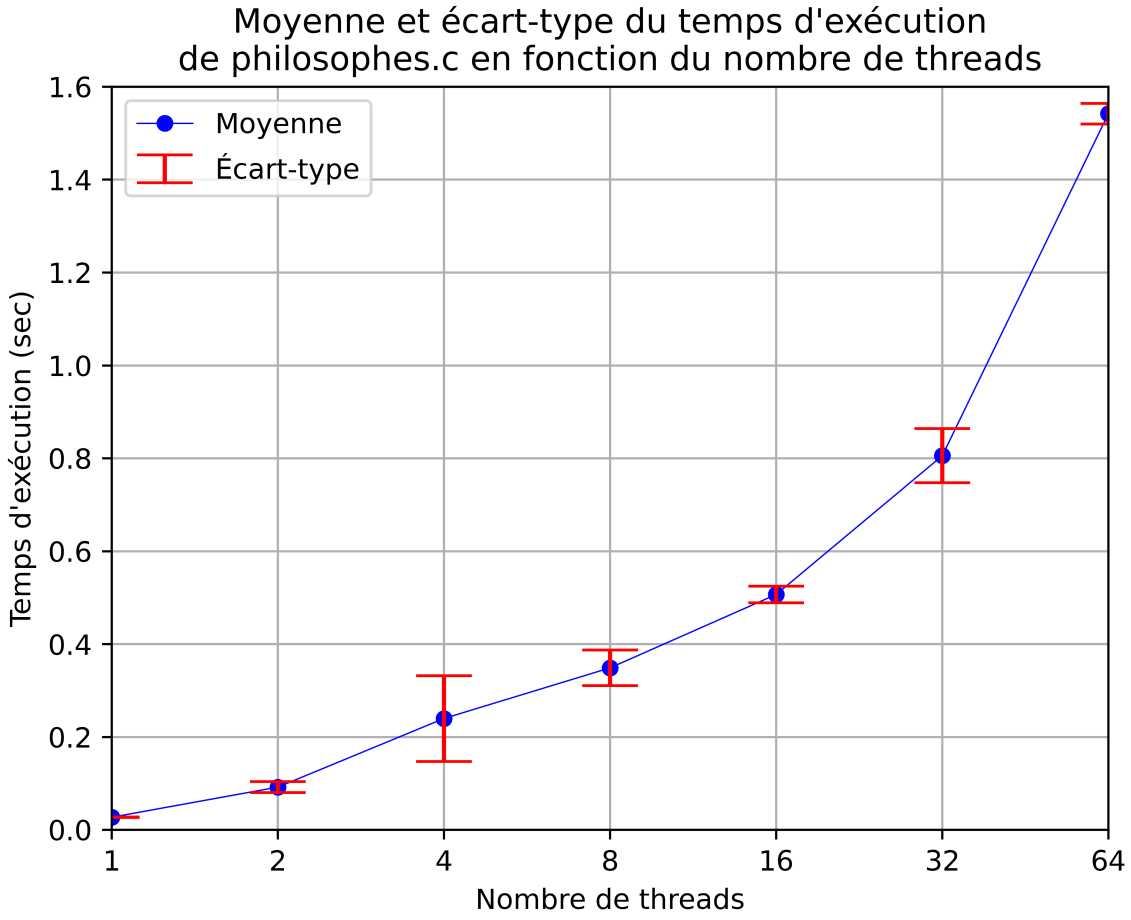

FIGURE 1 – Résultats pour philosophes.c avec 100.000 cycles penser/manger

Comme on peut le constater, au plus il y a de philosophes/threads, au plus il y a de baguettes/ressources communes ce qui signifie que les threads devront attendre plus longtemps pour avoir droit aux baguettes. Ajouter des threads dans ce modèle n'est pas beaucoup plus efficace puisque le partage des ressources est de plus en plus compliqué et prends donc un temps croissant d'exécution.

2.2 Problème des producteurs-consommateurs

Le problème des producteurs-consommateurs est un modèle couramment utilisé lorsque l'on doit réaliser de longs calculs. Deux types de threads vont se partager le travail :

— les threads producteurs : ces threads exécutent les calculs et placent leurs résultats dans un buffer qui est une zone mémoire auxquels à la fois les producteurs et les consommateurs ont accès.

— les threads consommateurs : ces threads vont lire les résultats placés dans le buffer et les utiliser.

L'idée de ce modèle est que les consommateurs puissent avancer à leur rythme sans que les producteurs ne bloquent inutilement les consommateurs et inversement.

2.2.1 Résultats et explications

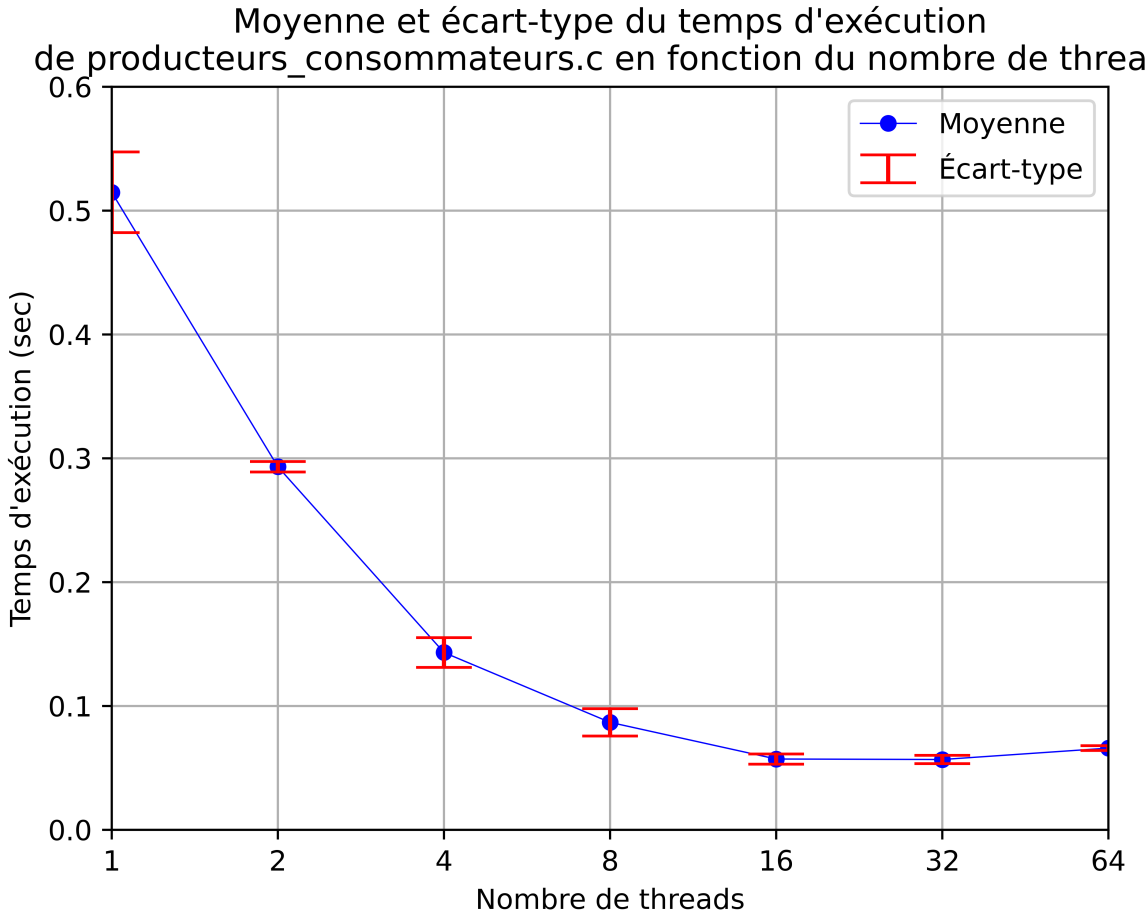

FIGURE 2 – Résultats pour producteurs_consommateurs.c avec un buffer de 8 places et 8192 éléments produits/consommés

Les résultats sont très clairs quand à ce modèle : avec environ 0.5 secondes de temps d'exécution en ayant qu'un seul producteur et consommateur il est rapide et l'ajout de threads le rends encore plus efficace. À partir de 16 threads le gain de temps devient négligeable et est sub-0.1 secondes.

2.3 Problème des lecteurs-écrivains

Le problème des lecteurs-écrivains est un modèle utilisé lorsque les threads doivent accéder à une base de données.
Une fois de plus, nous allons avoir deux différents types de threads :

— les lecteurs : ces threads permettent de lire une base de données. Plusieurs peuvent tourner en même temps.

— les écrivains : ces threads permettent de modifier une base de données. Si un écrivain écrit, aucun autre thread, qu'il soit écrivain ou lecteur ne peut travailler en même temps dessus.

2.3.1 Résultats et explications

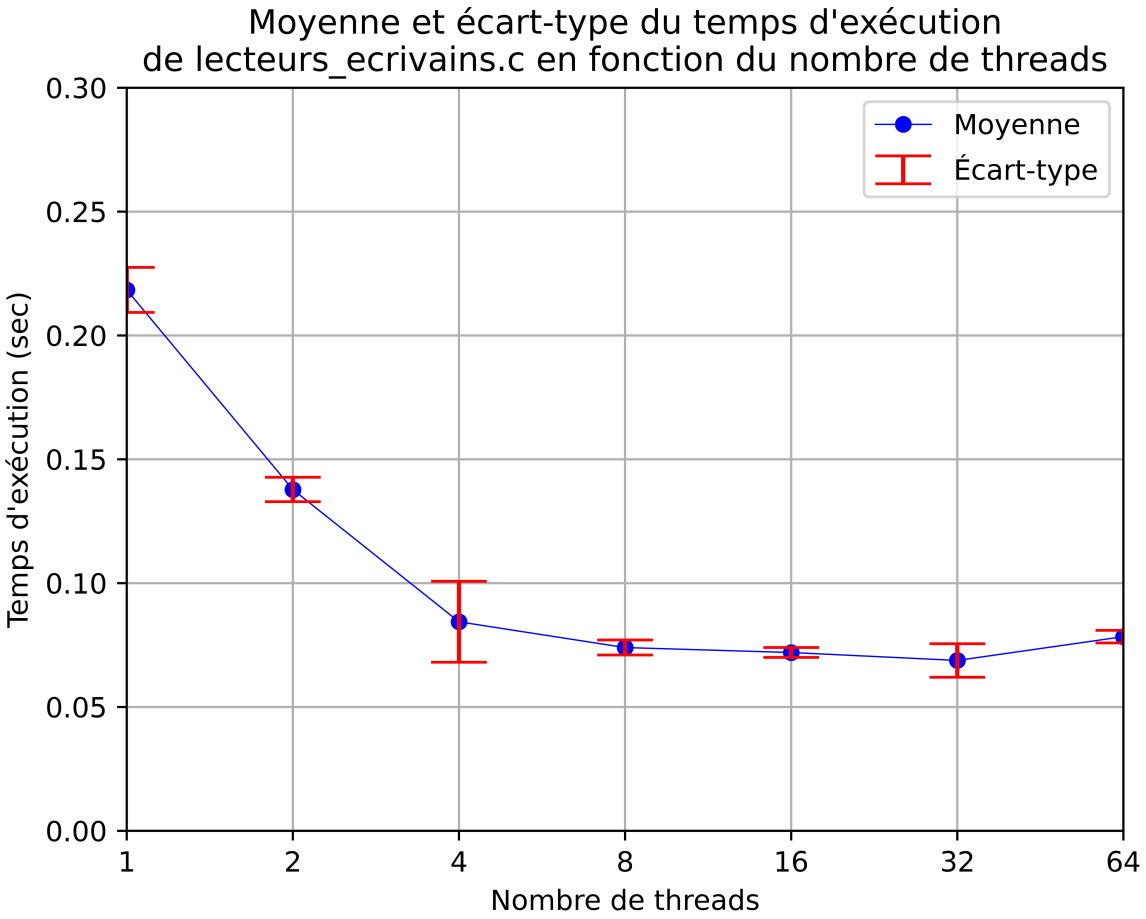

FIGURE 3 – Résultats pour lecteurs_ecrivains.c avec un nombre total de lectures et d'écritures respectivement de 2560 et 640

Le modèle est très rapide comparé au précédent. À partir de 8 threads, le gain de temps devient négligeable (le temps varie encore avec 4 threads comme on peut le constater avec l'écart-type) et finit par stagner à une durée sub-0.1 secondes.

3 Conclusion

Ce projet démontre parfaitement que plus de threads ne signifie pas forcément plus d'efficacité. Le modèle du programme est également important. Certains de ces modèles comme celui des philosophes devient par exemple moins efficace. D'autres deviendront plus rapide au plus il y a de threads mais avec une efficacité qui finira par stagner à partie d'un certain nombre de threads.
Il ne suffit pas de coder, il faut aussi faire un plan avant de passer à l'action !