UCLouvain

LINFO1252
Systèmes Informatiques
Rapport P1

Programmation multi-threadée et évaluation de performances

Élèves :
Justin Muylkens - 80042200
Diego Seisdedos Stoz - 6592300

Enseignant :
Étienne Rivière

7 décembre 2025

1 Partie 1 : Utilisation des primitives de synchronisation POSIX

1.1 Problème des philosophes

Nous observons à la figure 1 que le temps d'exécution évolue linéairement en fonction du nombre de threads.

Ceci est dû au fait que la création de nouveaux philosophes, représente une charge de travail de $1,000,000$ de cycles manger-penser. La machine physique ayant 16 coeurs et la création de thread consommant des ressources au processeur. En conclusion, cette évolution linéaire est liée à la nature du problème pour lequel chaque ajout de thread augmente le travail à effectuer.

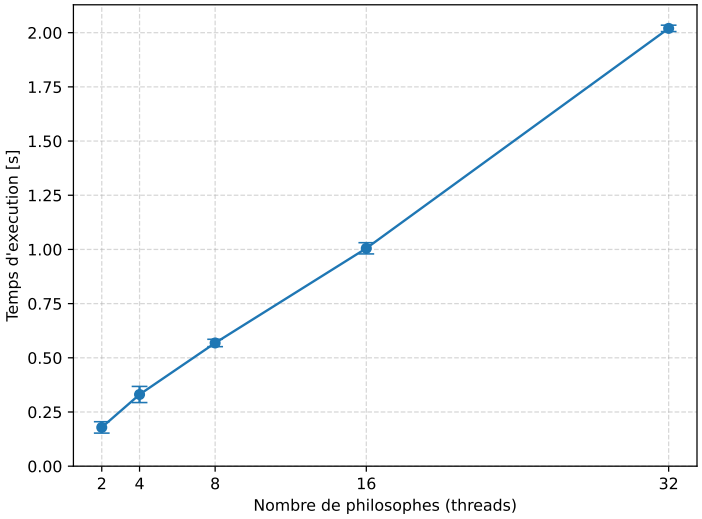

FIGURE 1 – Temps d'exécution du "problème des philosophes" en fonction du nombre de threads

1.2 Producteurs Consommateurs

La figure 2 montre un saut plus important du temps d'exécution entre 2 et 8 threads au total, autrement dit pour 1 à 4 producteurs/consommateurs.

Ce saut vient de la création de threads qui dépasse le gain d'efficacité sur la tâche de production et de consommation des threads. Ensuite, on atteint un plateau lié aux limites physiques de la machine.

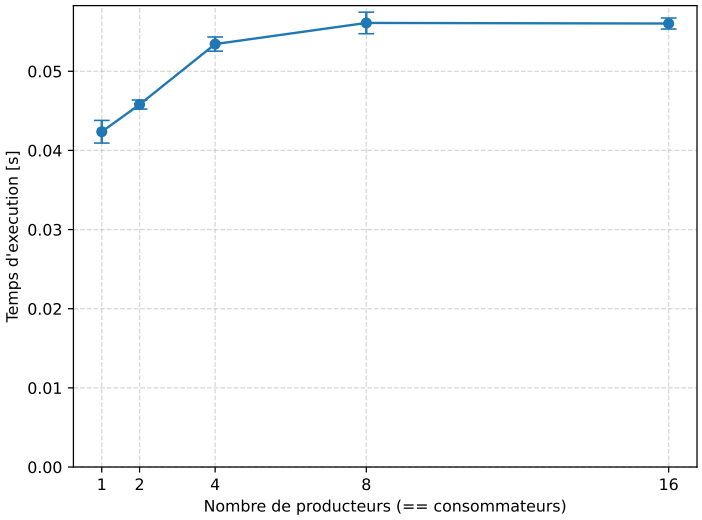

FIGURE 2 – Temps d'exécution du "producteurs-consommateurs" en fonction du nombre de threads

1.3 Lecteurs Écrivains

Le graphique de la figure 3 nous montre un comportement analogue d'explosion et de stabilisation du temps d'exécution.

Cela s'explique par la contention excessive des locks. Quand le nombre de threads augmente, les threads attendent de plus en plus sur les mêmes mutex. Dès lors, ils passent plus de temps à attendre qu'à travailler.

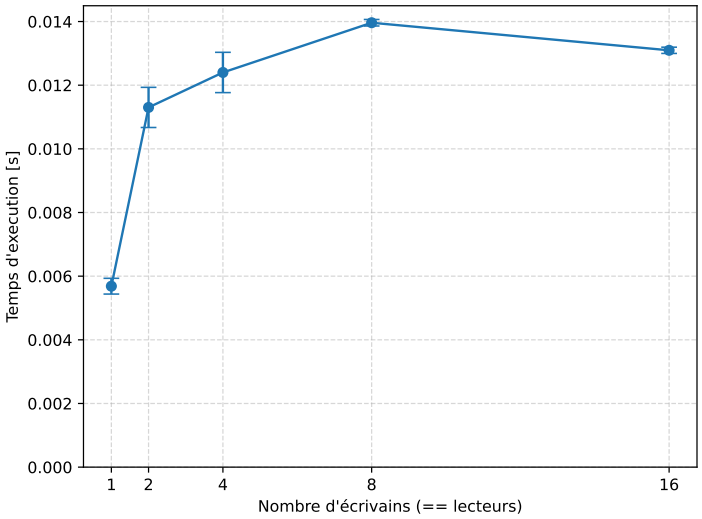

FIGURE 3 – Temps d'exécution du "lecteurs-écrivains" en fonction du nombre de threads

2 Partie 2 : Mise en oeuvre des primitives de synchronisation par attente active

2.1 Algorithme *test-and-set*

Le graphique 4 illustre les limites de l'approche *test-and-set* sous forte concurrence : alors que le temps d'exécution reste stable entre 1 et 4 threads (car la charge de travail globale est constante et simplement sérialisée), il se dégrade nettement à partir de 8 threads pour tripler presque à 32 threads.

Cette augmentation du temps est due à la contention excessive sur le bus mémoire : l'instruction atomique *xchg*, exécutée en boucle par tous les threads en attente active. Ceci sature le bus de communication et force l'invalidation des caches des processeurs, ce qui fait que le système passe plus de temps à gérer la cohérence mémoire qu'à exécuter les sections critiques.

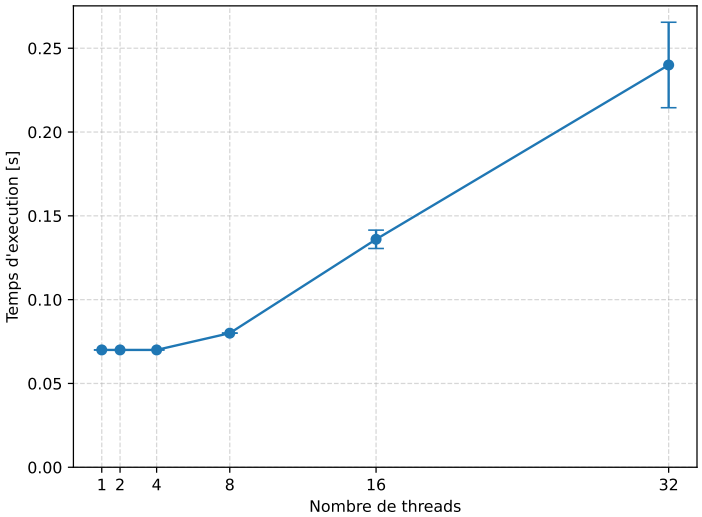

FIGURE 4 – Temps d'exécution de l'algorithme *test-and-set* en fonction du nombre de threads

2.2 Algorithme *test-and-test-and-set*

Le graphique de la figure 5 montre que l'agorithme TASTAS *test-and-test-and-set* est plus efficace que le TAS classique pour un nombre élevé de thread : alors que les deux méthodes offrent des performances similaire pour un petit nombre de thread, le TTAS limite considérablement la dégradation des performances observée à 16 et 32 threads.

Cette amélioration s'explique par la réduction du trafic sur le bus mémoire : contrairement au TAS qui force une instruction atomique coûteuse à chaque cycle d'attente, le TTAS permet aux threads de boucler en lecture sur leur cache local, ne déclenchant une tentative d'écriture atomique sur le bus que lorsque le verrou est détecté comme libre. De ce fait, la saturation du bus diminue ainsi que les invalidations de cache intempestives.

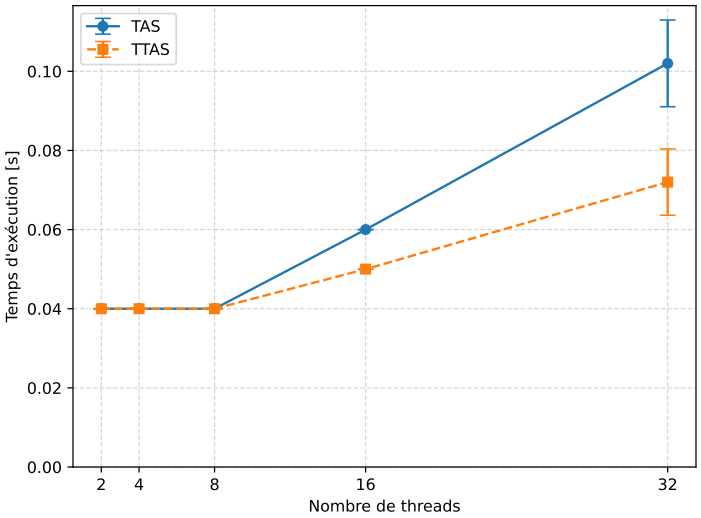

FIGURE 5 – Temps d'exécution de l'algorithme *test-and-test-and-set* en fonction du nombre de threads (sur machine perso)

3 Conclusion

Ce projet nous a permis de mieux comprendre les mécanismes de synchronisation entre threads et les problèmes liés au partage de ressources.

En implémentant les philosophes, le producteur-consommateur et les lecteurs-écrivains, nous avons vu l'importance des mutex et des sémaphores pour éviter les interblocages et les conditions de concurrence. La partie sur les verrous TAS et TATAS nous a montré l'impact des opérations atomiques sur les performances, et aussi l'intérêt d'optimiser l'attente active.

Enfin, la réalisation des mesures et des graphes nous a appris à analyser expérimentalement le comportement des programmes parallèles. Ce projet nous a donc permis de relier la théorie à la pratique de manière concrète.