UCLouvain

ÉCOLE POLYTECHNIQUE DE LOUVAIN

LINFO1252 - Systèmes Informatiques

Projet 1 : Programmation multi-threadée et évaluation de performances

Groupe 19

Décembre 2023

Abarkan Rayan (29712300)
rayan.abarkan@student.uclouvain.be
Niyikiza Cédric (64492300)
cedric.niyikiza.@student.uclouvain.be

Professor : Riviere Etienne

1 Analyses des expérimentations

1.1 Philosophes

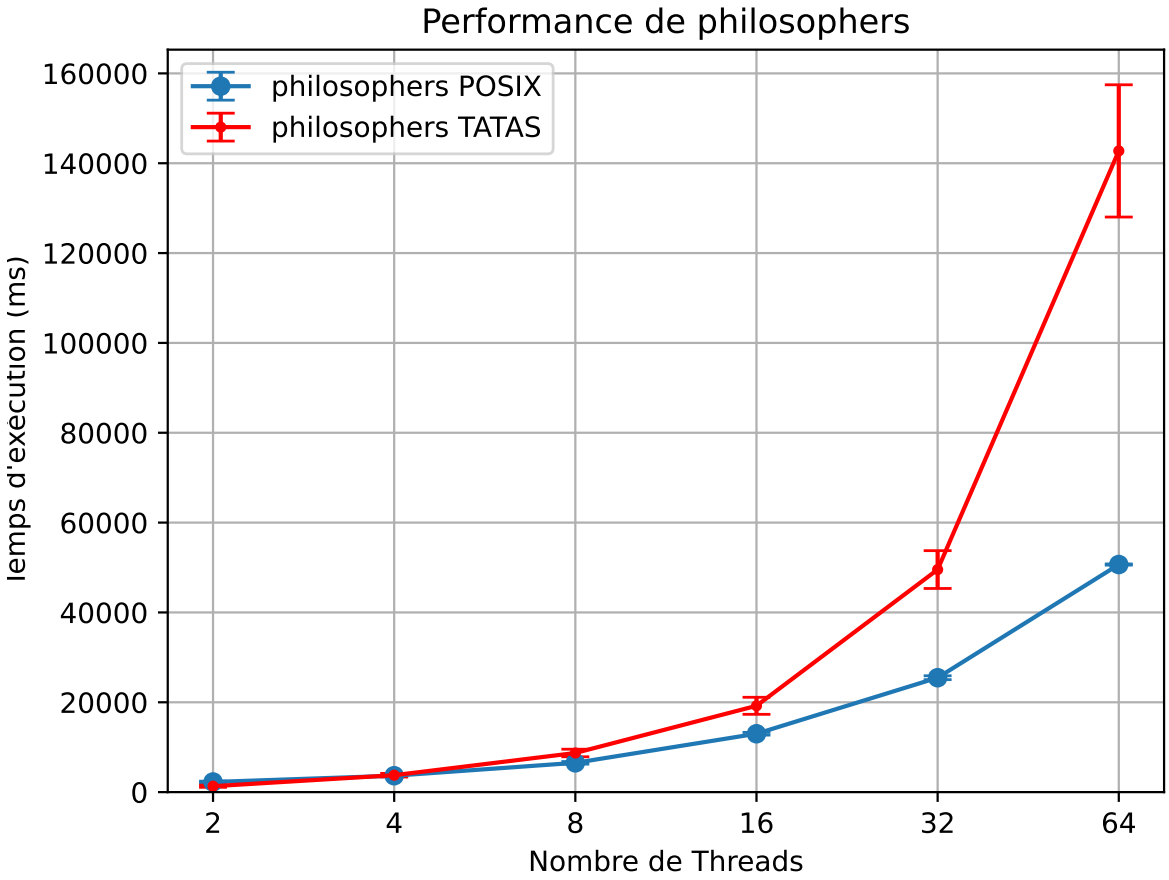

FIGURE 1 – Évolution du temps d'exécution en fonction du nombre de threads : une courbe pour la version utilisant la librairie POSIX (mutex et/ou semaphores) et une seconde courbe avec une implémentation personnalisée utilisant des verrous d'attentes active.

L'analyse comparative des mutex POSIX et de notre implémentation TATAS [1] pour le problème des philosophes a révélé des aspects intéressants. Les mutex POSIX, issus d'une bibliothèque standard, se montrent robustes et polyvalents, adaptés à divers contextes de multi-threading. D'un autre côté, notre approche TATAS excelle dans des situations à faible contention, avec son modèle d'attente active efficace pour des sections critiques brèves. Cependant, en cas de forte contention ou de sections critiques prolongées, les mutex POSIX semblent offrir de meilleures performances, soulignant ainsi l'importance du choix de la stratégie de synchronisation adaptée à chaque situation.

1. TATAS fait référence à l'implémentation de nos verrous avec l'algorithme Test-And-Test-And-Set

1.2 Producteurs-Consommateurs

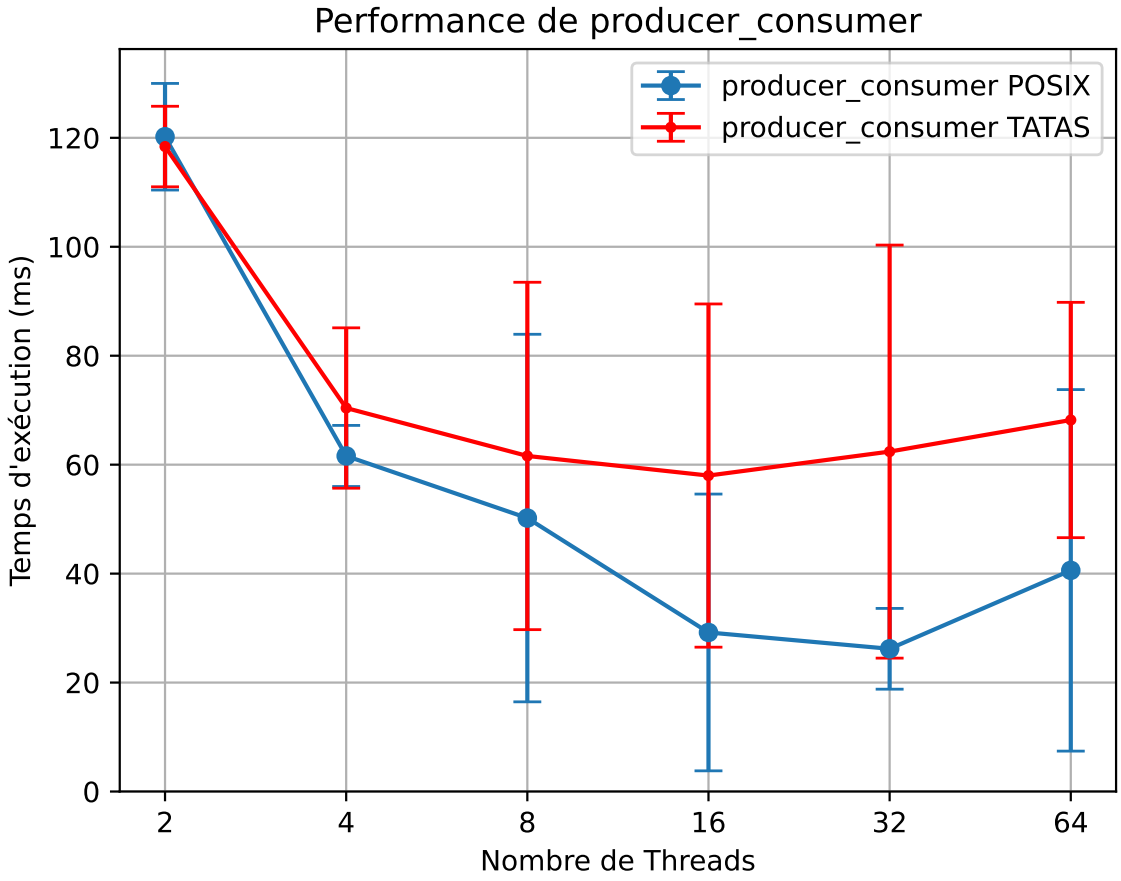

FIGURE 2 – Évolution du temps d'exécution en fonction du nombre de threads : une courbe pour la version utilisant la librairie POSIX (mutex et/ou semaphores) et une seconde courbe avec une implémentation personnalisée utilisant des verrous d'attentes active.

L'analyse des performances pour les programmes producteur-consommateur montre une augmentation de l'écart type des temps d'exécution avec le nombre de threads. Cela indique que la fiabilité des performances diminue à mesure que le nombre de threads augmente, probablement en raison d'une contention plus élevée et d'une gestion plus complexe des ressources. Cette variabilité accrue avec l'augmentation des threads souligne l'importance de choisir le bon niveau de parallélisme pour équilibrer efficacement l'utilisation des ressources tout en maintenant une régularité dans les performances.

1.3 Lecteurs et écrivains

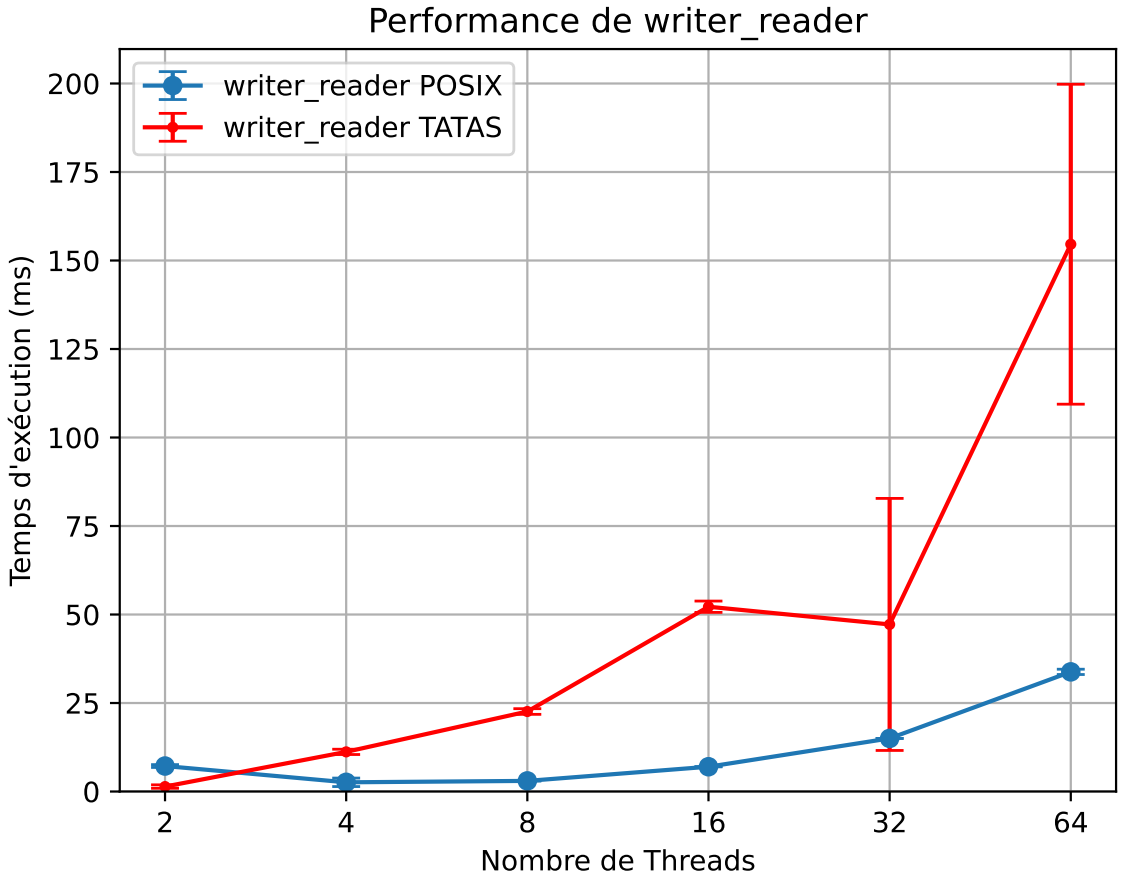

FIGURE 3 – Évolution du temps d'exécution en fonction du nombre de threads : une courbe pour la version utilisant la librairie POSIX (mutex et/ou semaphores) et une seconde courbe avec une implémentation personnalisée utilisant des verrous d'attentes active.

1.4 Test-And-Set & Test-And-Test-And-Set

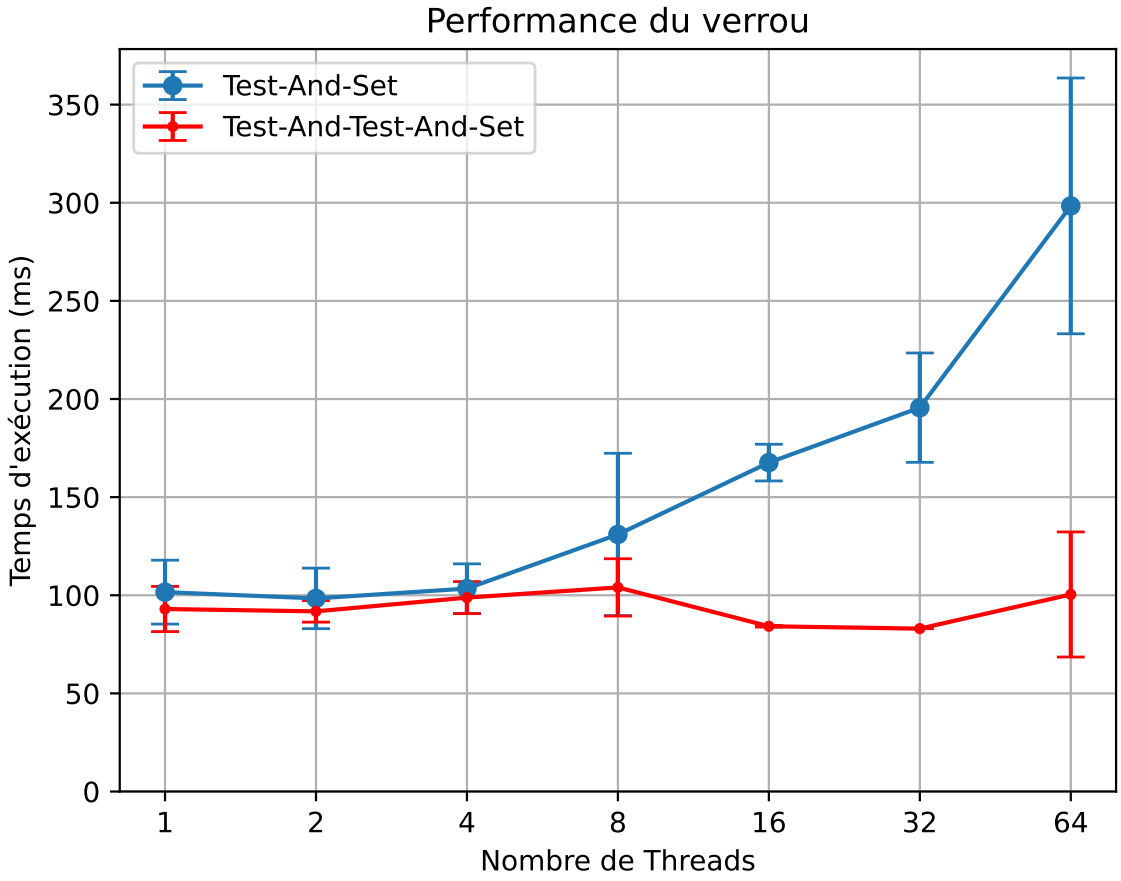

FIGURE 4 – Évolution du temps d'exécution en fonction du nombre de threads : une courbe pour la version utilisant l'algorithme ***Test-And-Set*** et une seconde courbe pour l'utilisation du ***Test-And-Test-And-Set***.

2 Conclusion

note : - variance sur notre implémentation - diminution de l'ecart type avec l'implementation du spin dans le test and set == tatas - bien fait du tatas - -