UCLouvain
École polytechnique de Louvain

LINFO1252 - Systèmes Informatiques

-

Rapport

Programmation multi-threadée et évaluation de performances

Étudiants (groupe50-mercredi):

63401800 - Bousmar Cyril

57961800 - Kafrouni Christophe

Professeur :

Riviere Etienne

December 10, 2023

Introduction

Ce rapport fait état de notre évaluation des performances d'applications en langage `C` utilisant plusisieurs threads et des primitives de synchronization.

Les primitives utilisées sont les mutex, les sémaphores et des verrous avec attente active des librairies standard de `C` et de notre propre implémentation.

Les applications évalués représentes les 3 problèmes suivants : le problèmes des philosophes[1], le problèmes des producteurs/consommateurs[2] et le problèmes des lecteurs/écrivains[3].

Environnement

L'évaluation des perfomances a été conduite sous *Windows*, avec un processeur `AMD Ryzen 7 5800H @3.2-4.4GHz`[4] possèdant 8 `CPUs` et 16 coeurs logiques. Les caches `L1`, `L2` et `L3` ont respectivement `256 KiB`, `4 MiB` et `16 MiB`.

[1] O.Bonaventure, E.Riviere, G.Detal, C.Paasch.. *Le problème des philosophes.* `https://sites.uclouvain.be/SystInfo/notes/Theorie/Threads/threads2.html#le-probleme-des-philosophes`. (visité le 05/11/2023).

[2] O.Bonaventure, E.Riviere, G.Detal, C.Paasch.. *Problème des producteurs-consommateurs.* `https://sites.uclouvain.be/SystInfo/notes/Theorie/Threads/coordination.html#probleme-des-producteurs-consommateurs`. (visité le 05/11/2023).

[3] O.Bonaventure, E.Riviere, G.Detal, C.Paasch.. *Problème des readers-writers.* `https://sites.uclouvain.be/SystInfo/notes/Theorie/Threads/coordination.html#probleme-des-readers-writers`. (visité le 05/11/2023).

[4] AMD. *AMD Ryzen™ 7 5800H Mobile Processor.* `https://www.amd.com/en/products/apu/amd-ryzen-7-5800h` (visité le 10/12/2023).

Évaluation des performances

Verrous

Sans grande surprise, nous pouvons constater (cf. figure 0-1) que l'utilisation des verrous, quels qu'ils soient, est relativement rapide.

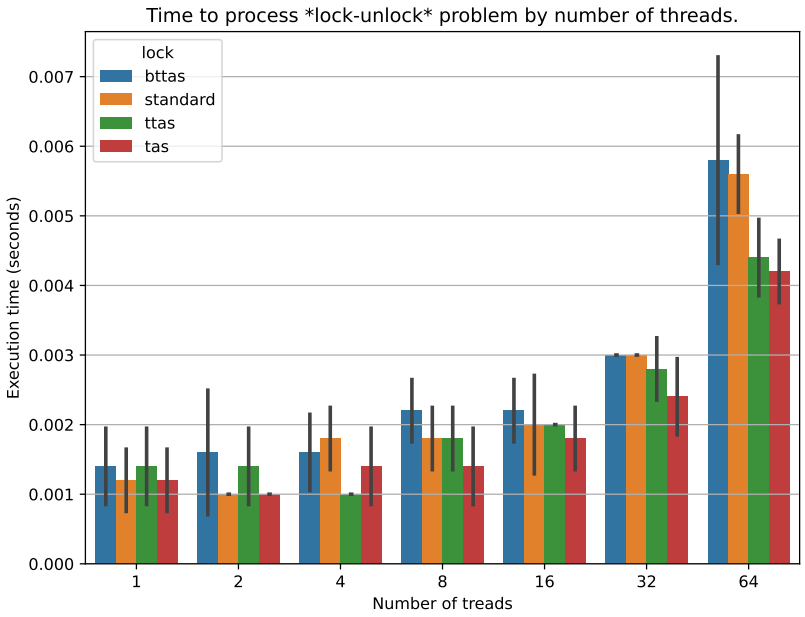

Figure 0-1: Temps d'éxécution pour l'application `lock-unlock.c` par nombre de thread.

Étant donné que l'application ne fait qu'éxécuter des cycles répétés de `lock` et `unlock` avec une section critique vide, il est intéressant de constater que le `TAS` est le plus rapide. En effet, il n'utilise pas de tests supplémentaires comme le `TTAS`. Enfin, le `BTTAS` est le plus lent étant donné qu'en plus d'effectuer le test du `TTAS`, il met les threads en veille pendant une durée qui grandi exponentiellement.

Prolème des philosophes

Cette application est celle qui, de loin, a le temps d'éxécution le plus long.

Il est intéressant de constater deux choses (cf. figure 0-2).

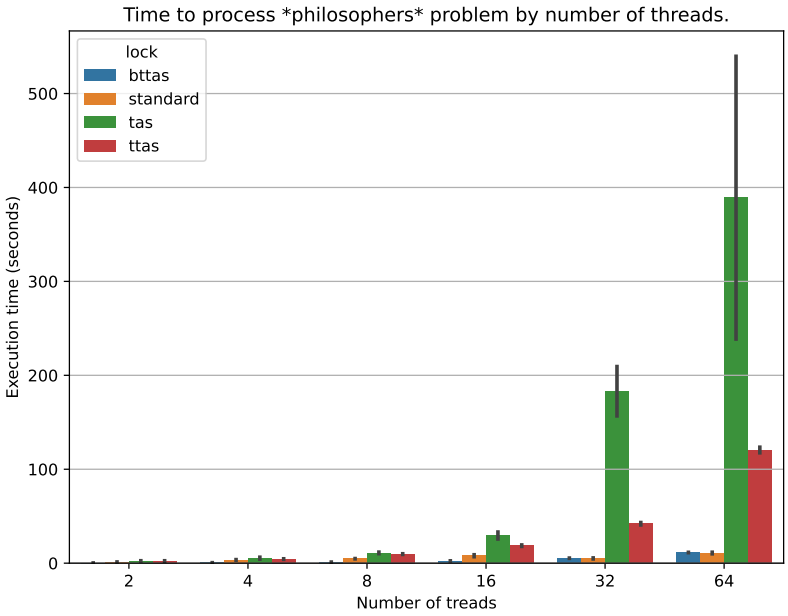

Figure 0-2: Temps d'éxécution pour l'application `philosophers.c` par nombre de thread.

D'une part, le `TAS` devient très lent (et inutilisable) au plus le nombre de threads augmente, alors que le `TTAS` n'est qu'une amélioration de ce premier. C'est un exemple parfait de la saturation du `BUS`. Le premier algorithme entrainant une saturation constante alors que le suivant entraine une saturation ponctuelle, à chaque libération de la ressource. Au plus il y a de threads, au plus l'impact est grand, car tous essaient d'accéder concourament à la ressource.

D'autre part, le `BTTAS` est aussi performant que le verrou standard. Ces deux-ci implémentant des comportements de contention en désynchronisant les moments lors desquels les threads essaieront d'accéder à la ressource, pour éviter des pics d'affluence.

Problèmes des producteurs-consommateurs

On constate que `BTTAS` possède une performance peu prédictible dans ce problème. Néamoins à faible nombre de threads, les verrous `TAS` et `TTAS` performent exceptionellement bien, mais celà ne dure pas, pour un plus grand nombre de thread (donc de production et consommations) ces verrous bloqueront le bus beaucoup plus souvent inutilement.

Ce problème grandissant linéairement, la performance du verrous standard est assez stable.

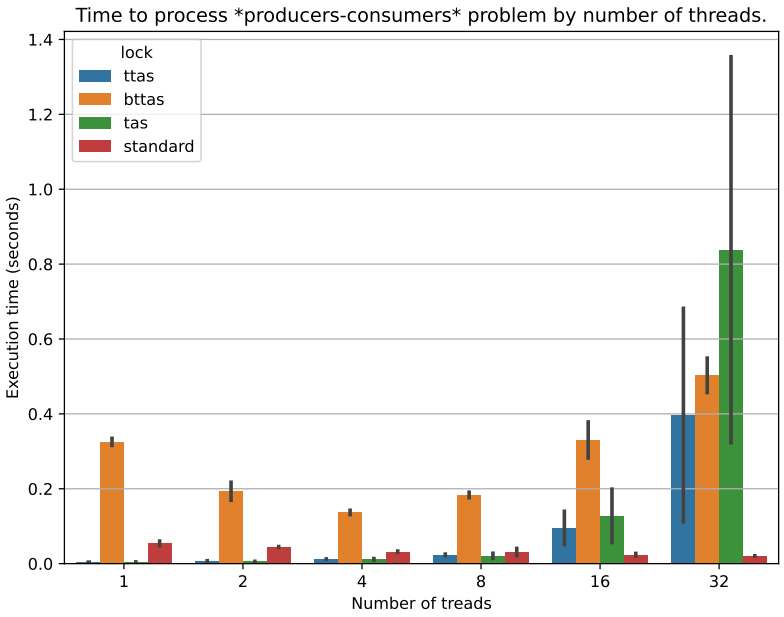

Figure 0-3: Temps d'éxécution pour l'application `producers-consumers.c` par nombre de thread.

Problèmes des lecteurs-écrivains

On constate que les verrous standard et `BTTAS` performent très bien pour ce problème à raison que beaucoup de thread seront bloqué pour un certain temps vu que le problème bloque toutes les threads lecteur, si uniquement un thread écrivain tourne. Donc mettre le reste des threads en attente passive est une meilleur approche au lieu de les laissé bloqué le bus inutilement.

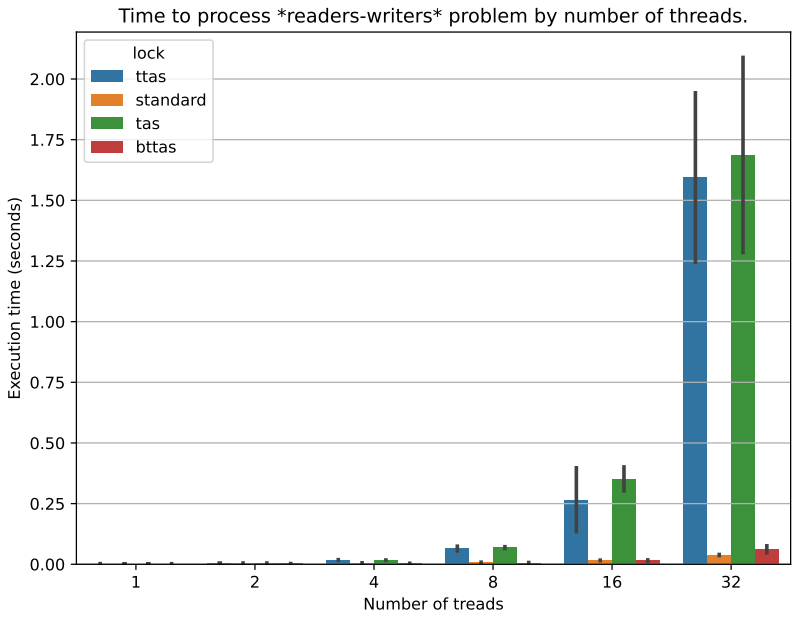

Figure 0-4: Temps d'éxécution pour l'application `readers-writers.c` par nombre de thread.

Conclusion

En conclusion, nous sommes confiants quant à l'exactitude de notre implémentation étant donné que les résultats convergent et appuient nos suppositions théoriques.

Enfin, après avoir passé beaucoup de temps à optimiser les différents verrous, nous nous souviendrons de l'impact du processeur, de son architecture et de son environnement sur les résultats obtenus. Ceux-ci peuvent en effets varier grandement.