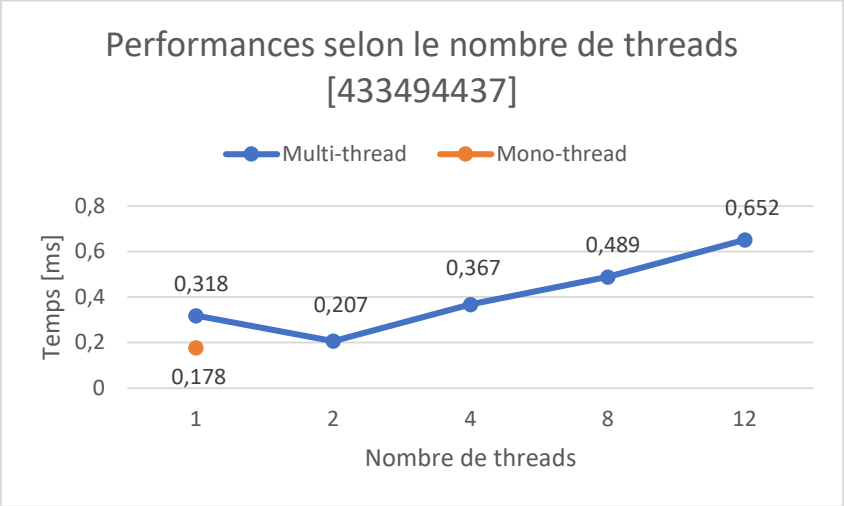
1 ANALYSE DES PERFORMANCES

Nous allons utilisé les 4 exemples ci-dessous afin de comparer les performances entre les exécutions mono-thread et multi-thread. Pour les exécutions à plusieurs threads, nous avons différents résultats selon le nombre de threads qui était utilisés.

Dans notre 4 exemples, on retrouve une paire de petit nombre et une paire de nombre assez grands. Chaque paire étant composée d'un nombre premier et d'un nombre composé.

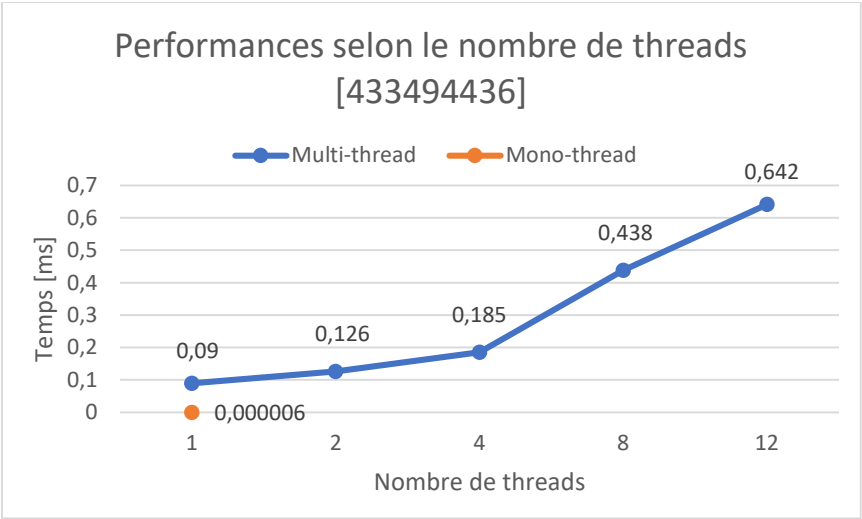
1.1 Exemple 1 [433494437]

Ici, il n'est pas vraiment avantageux d'augmenter le nombre de threads. L'architecture mono-thread est plus efficace que n'importe quelle exécution multi-thread. De plus, on remarque que plus le nombre de threads augmente, plus on prend de temps à trouver le résultat. On peut expliquer cela via le fait que c'est un petit nombre. Plus la tâche est triviale et rapide à exécuter, plus le fait de créer un thread prend du temps par rapport au temps que l'on aurait mis à résoudre le problème directement.



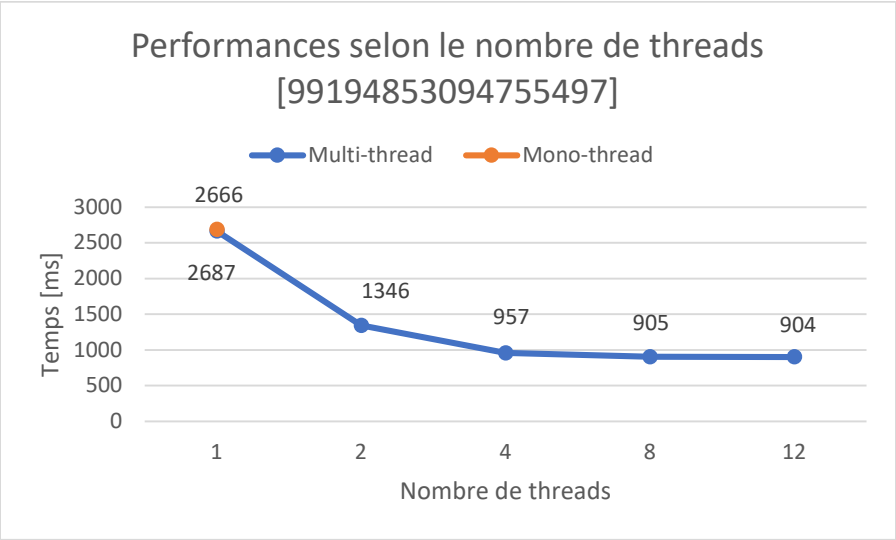
1.2 Exemple 2 [433494436]

Comme pour le nombre précédent, dû à son échelle il n'est pas avantageux ici d'augmenter le nombre de threads. De plus, ce nombre a la particularité d'être paire. Donc dans une architecture mono-thread, il sera automatiquement détecté en tant que nombre composé dès la première itération. Ce qui finalement rend le multi-thread encore mon viable par rapport à du mono-thread.



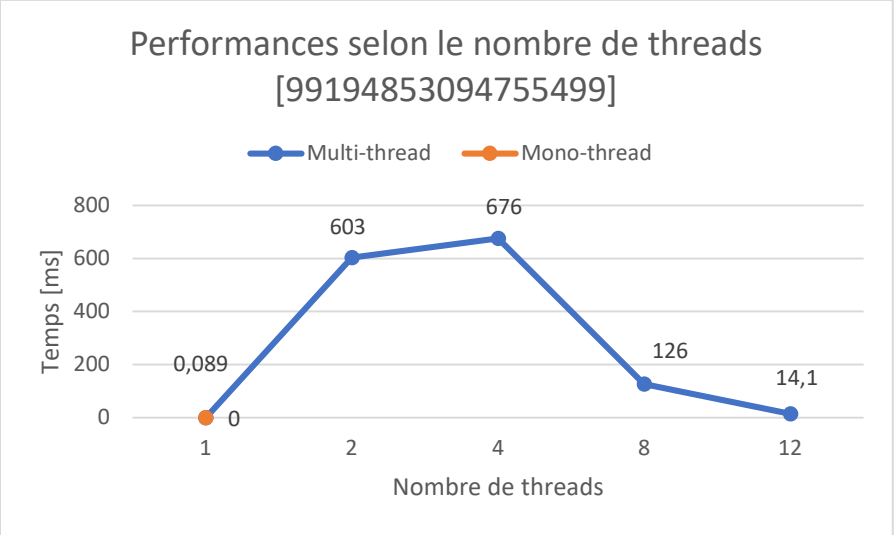
1.3 Exemple 3 [99194853094755497]

Pour un grand nombre premier, on remarque tout de suite que le multi-thread est avantageux. On réduit le temps de plus de moitié lorsque l'on augmente le nombre de threads. On remarque cependant que l'amélioration du temps n'est pas sans fin, à partir d'un certain nombre de threads en parallèle la courbe s'aplatie et le facteur de diminution n'est plus aussi grand.



1.4 Exemple 4 [99194853094755499]

Ici c'est un grand nombre, cependant il n'est pas premier. On remarque alors que le mono-thread est assez efficace par rapport au multi-thread. On peut aussi voir que lorsque l'on exécute le programme en multi-thread les performances s'améliorent en augmentant le nombre de threads. Il y a cependant une donnée assez incohérente avec 4 threads qui est toujours la valeur la plus élevée (avec beaucoup de marge selon les tests).



2 SYNTHÈSE

Ce que l'on remarque directement, c'est qu'il n'est pas forcément avantageux d'utiliser du multi-thread pour des petits nombres qu'ils soient premiers ou composés. La création et gestion des threads prend plus de temps que simplement déterminer la primalité du nombre.

On voit aussi que le multi-thread est très intéressant pour les nombres premiers, la courbe de progression est bonne et on réduit considérablement le temps de calcul. Tandis que pour les nombres composés, il peut s'avérer intéressant d'avoir plusieurs threads, mais le résultat diverge plus que pour un nombre premier.

Le gain de vitesse est assez linéaire pour un nombre grand et premier, mais il est assez irrégulier lors des autres combinaisons que l'on a pu tester.

2.1 Spécifications techniques

Ci-dessous se trouvent les spécifications techniques avec lesquelles ont été menés les tests. La machine hôte étant sous la distribution Ubuntu avec la version 22.04.

```
C:\home\alexis> lscpu
Architecture:            x86_64
  CPU op-mode(s):        32-bit, 64-bit
  Address sizes:         39 bits physical, 48 bits virtual
  Byte Order:            Little Endian
CPU(s):                  8
  On-line CPU(s) list:   0-7
Vendor ID:               GenuineIntel
  Model name:            11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80GHz
    CPU family:          6
    Model:               140
    Thread(s) per core:  2
    Core(s) per socket:  4
    Socket(s):           1
    Stepping:            1
    CPU max MHz:         2800.0000
    CPU min MHz:         400.0000
```