

RAPPORT DU TP3 : SYSTÈMES CONCURRENTS

Parallèles



MYRIAM ROBBANA

DECEMBRE 2024

Table des matières

1	Calcul du maximum d'un tableau.	3
1.1	Comparaison des performances des solutions.	3
1.2	Etude de la variation du nombre de threads, de la taille du pool et du seuil d'arrêt. . . .	4
1.3	Analyser les résultats	5
2	Comptage des éléments dans un intervalle.	5
2.1	Comparaison des performances des solutions.	5
2.2	Etude de la variation du nombre de threads, de la taille du pool et du seuil d'arrêt. . . .	7
2.3	Analyser les résultats	8

Table des figures

1	java MaxTabSequential foo 20	3
2	java MaxTabThread foo 20 4	3
3	java MaxTabPool foo 20 4 8	3
4	java MaxTabForkJoin foo 20 1000	3
5	java CountSequential foo 20	6
6	java CountThread foo 20 4	6
7	java CountPool foo 20 4 8	6
8	java CountForkJoin foo 20 1000	6

1 Calcul du maximum d'un tableau.

Le but est de trouver l'élément maximal du tableau.

1.1 Comparaison des performances des solutions.

```
Myriam@archlinux: ~/Documents/N7/2A/systemes_concurrents/TP3/TP-Parallele/max$ java MaxTabSequential foo 20
10 processeurs disponibles pour la JVM
Essai [0] : result = 1000, durée 3012 µs
Essai [1] : result = 1000, durée 633 µs
Essai [2] : result = 1000, durée 2044 µs
Essai [3] : result = 1000, durée 2229 µs
Essai [4] : result = 1000, durée 2111 µs
Essai [5] : result = 1000, durée 2034 µs
Essai [6] : result = 1000, durée 548 µs
Essai [7] : result = 1000, durée 324 µs
Essai [8] : result = 1000, durée 520 µs
Essai [9] : result = 1000, durée 520 µs
Essai [10] : result = 1000, durée 520 µs
Essai [11] : result = 1000, durée 537 µs
Essai [12] : result = 1000, durée 535 µs
Essai [13] : result = 1000, durée 508 µs
Essai [14] : result = 1000, durée 502 µs
Essai [15] : result = 1000, durée 502 µs
Essai [16] : result = 1000, durée 502 µs
Essai [17] : result = 1000, durée 502 µs
Essai [18] : result = 1000, durée 508 µs
Essai [19] : result = 1000, durée 427 µs
Moyenne des durées (4 premiers essais ignorés) = 799 µs
```

FIGURE 1 – java MaxTabSequential foo 20

```
Myriam@archlinux: ~/Documents/N7/2A/systemes_concurrents/TP3/TP-Parallele/max$ java MaxTabThread foo 20 4
10 processeurs disponibles pour la JVM
Essai [0] : result = 1000, durée 37192 µs
Essai [1] : result = 1000, durée 1026 µs
Essai [2] : result = 1000, durée 1628 µs
Essai [3] : result = 1000, durée 2085 µs
Essai [4] : result = 1000, durée 2097 µs
Essai [5] : result = 1000, durée 963 µs
Essai [6] : result = 1000, durée 533 µs
Essai [7] : result = 1000, durée 732 µs
Essai [8] : result = 1000, durée 885 µs
Essai [9] : result = 1000, durée 805 µs
Essai [10] : result = 1000, durée 986 µs
Essai [11] : result = 1000, durée 639 µs
Essai [12] : result = 1000, durée 3581 µs
Essai [13] : result = 1000, durée 764 µs
Essai [14] : result = 1000, durée 671 µs
Essai [15] : result = 1000, durée 492 µs
Essai [16] : result = 1000, durée 467 µs
Essai [17] : result = 1000, durée 482 µs
Essai [18] : result = 1000, durée 625 µs
Essai [19] : result = 1000, durée 499 µs
Moyenne des durées (4 premiers essais ignorés) = 944 µs
```

FIGURE 2 – java MaxTabThread foo 20 4

```
Myriam@archlinux: ~/Documents/N7/2A/systemes_concurrents/TP3/TP-Parallele/max$ java MaxTabPool foo 20 4 8
10 processeurs disponibles pour la JVM
Essai [0] : result = 1000, durée 32693 µs
Essai [1] : result = 1000, durée 986 µs
Essai [2] : result = 1000, durée 375 µs
Essai [3] : result = 1000, durée 364 µs
Essai [4] : result = 1000, durée 368 µs
Essai [5] : result = 1000, durée 319 µs
Essai [6] : result = 1000, durée 453 µs
Essai [7] : result = 1000, durée 342 µs
Essai [8] : result = 1000, durée 272 µs
Essai [9] : result = 1000, durée 292 µs
Essai [10] : result = 1000, durée 301 µs
Essai [11] : result = 1000, durée 272 µs
Essai [12] : result = 1000, durée 274 µs
Essai [13] : result = 1000, durée 331 µs
Essai [14] : result = 1000, durée 257 µs
Essai [15] : result = 1000, durée 313 µs
Essai [16] : result = 1000, durée 337 µs
Essai [17] : result = 1000, durée 341 µs
Essai [18] : result = 1000, durée 430 µs
Essai [19] : result = 1000, durée 277 µs
Moyenne des durées (4 premiers essais ignorés) = 324 µs
```

FIGURE 3 – java MaxTabPool foo 20 4 8

```
Myriam@archlinux: ~/Documents/N7/2A/systemes_concurrents/TP3/TP-Parallele/max$ java MaxTabForkJoin foo 20 1000
10 processeurs disponibles pour la JVM
Essai [0] : result = 1000, durée 35986 µs
Essai [1] : result = 1000, durée 840 µs
Essai [2] : result = 1000, durée 836 µs
Essai [3] : result = 1000, durée 843 µs
Essai [4] : result = 1000, durée 790 µs
Essai [5] : result = 1000, durée 900 µs
Essai [6] : result = 1000, durée 1100 µs
Essai [7] : result = 1000, durée 788 µs
Essai [8] : result = 1000, durée 661 µs
Essai [9] : result = 1000, durée 829 µs
Essai [10] : result = 1000, durée 1058 µs
Essai [11] : result = 1000, durée 727 µs
Essai [12] : result = 1000, durée 1241 µs
Essai [13] : result = 1000, durée 1124 µs
Essai [14] : result = 1000, durée 1017 µs
Essai [15] : result = 1000, durée 702 µs
Essai [16] : result = 1000, durée 726 µs
Essai [17] : result = 1000, durée 690 µs
Essai [18] : result = 1000, durée 641 µs
Essai [19] : result = 1000, durée 761 µs
Moyenne des durées (4 premiers essais ignorés) = 863 µs
```

FIGURE 4 – java MaxTabForkJoin foo 20 1000

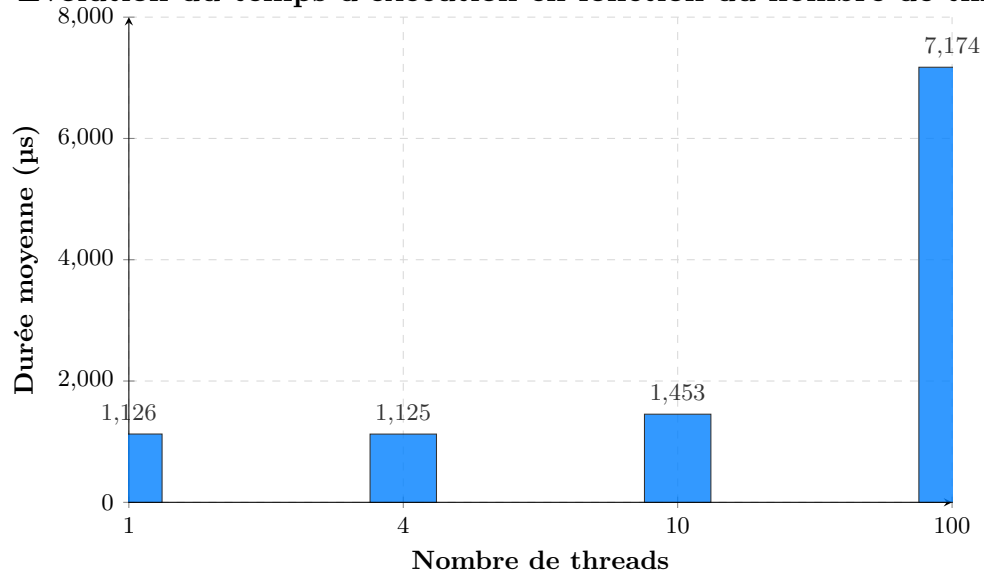
Méthode	Temps moyen (µs)	Remarque
MaxTabSequential	709	Séquentiel simple
MaxTabThread (4 threads)	944	Threads multiples
MaxTabPool (4 threads, 8 tâches)	324	Pool de threads
MaxTabForkJoin (1000 tâches)	863	Fork-Join

TABLE 1 – Comparaison des performances des différentes méthodes

On remarque que le résultat est bien cohérent avec le tableau créé, et que le maximum donné pour chacune des versions est bien égale à 1000. Cependant, la méthode *MaxTabPool*, avec un temps moyen de 324 µs, est la plus performante, tandis que la méthode *MaxTabThread*, avec un temps moyen de 944 µs, est moins efficace.

1.2 Etude de la variation du nombre de threads, de la taille du pool et du seuil d'arrêt.

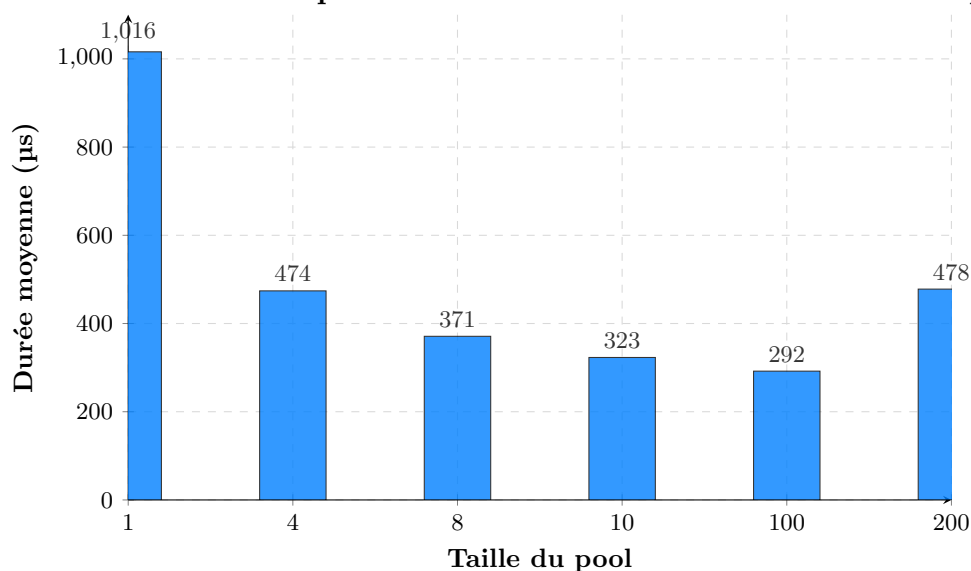
Evolution du temps d'exécution en fonction du nombre de threads



Logiquement, plus on augmente le nombre de threads, plus la durée d'exécution devrait diminuer. Cependant, au-delà d'un certain seuil, créer un trop grand nombre de threads devient contre-productif. En effet, le temps et les ressources nécessaires pour gérer ces threads dépassent les gains de parallélisme, rendant l'exécution plus longue que si l'on avait utilisé un nombre réduit de threads. Le graphe ci-dessus permet de mettre ce phénomène en exergue.

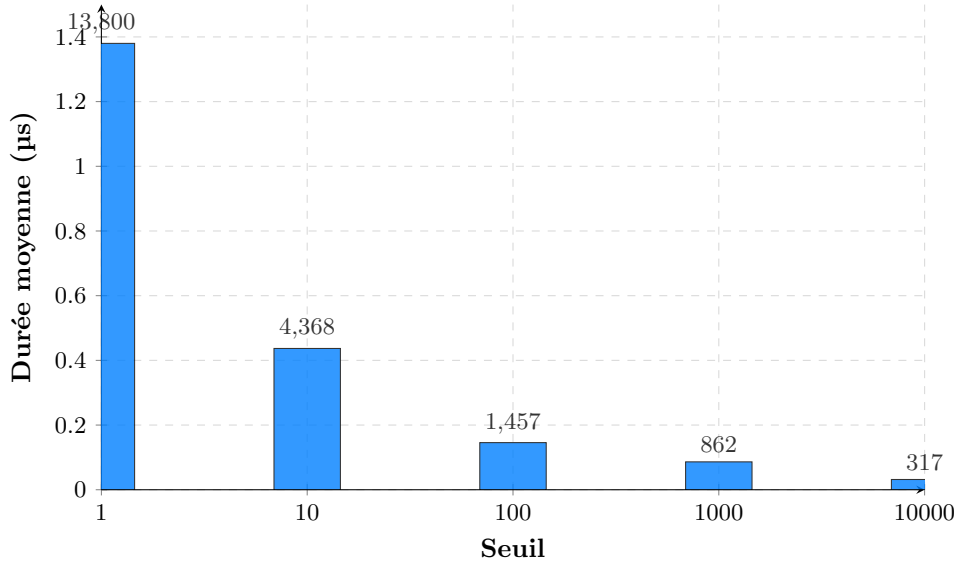
L'objectif est donc de trouver un équilibre : le nombre optimal de threads qui minimise la durée moyenne. Dans les exemples que j'ai réalisés, 4 threads semblent offrir un bon compromis !

Evolution du temps d'exécution en fonction de la taille du pool



On remarque que plus la taille de pool augmente et plus la durée d'exécution diminue jusqu'à un certain point. En effet, au-delà d'une taille de pool égale à 100, les performances diminuent légèrement, comme on peut le voir avec une augmentation du temps moyen pour une taille de pool égale à 200. Cela s'explique par le fait qu'augmenter la taille du pool au-delà d'une certaine limite n'apporte plus d'amélioration et peut même entraîner une légère perte d'efficacité. Le meilleur résultat dans notre exemple, est obtenu avec une taille de pool autour de 100, où le temps moyen d'exécution est le plus court.

Evolution du temps d'exécution en fonction de la valeur du seuil



On observe que plus le seuil est grand et plus le temps d'exécution diminue. Et pour un seuil assez élevé on atteint une durée moyenne d'exécution relativement rapide.

1.3 Analyser les résultats

L'optimisation du nombre de threads et de la taille du pool est importante pour améliorer les performances et diminuer la durée moyenne d'exécution. Bien qu'un nombre croissant de threads et une taille de pool plus grande réduisent le temps d'exécution, il existe un seuil au-delà duquel les gains deviennent négligeables, voire contre-productifs. Dans notre exemple, une taille de pool d'environ 100 et un nombre optimal de 4 threads offrent les meilleures performances, avec un temps d'exécution réduit de manière significative. QUand au seuil, plus ce dernier est élevé et plus l'exécution est rapide.

La version la plus efficace semble donc être celle qui prend en compte ces 3 paramètres dont le seuil pour minimiser un maximum la durée d'exécution. Cette version est `MaxTabForkJoin.java`

Les différentes comparaisons réalisées montrent les performance

Les fichiers fournis pour cette partie sont : `MaxTabThread.java`, `MaxTabPool.java`, `MaxTabForkJoin.java`.

2 Comptage des éléments dans un intervalle.

Le but est de compter le nombre de valeurs dans le tableau entre 0 et 9 (constantes `VMIN/VMAX` arbitrairement fixées).

2.1 Comparaison des performances des solutions.

```
Myriam@archLinux: ~/Documents/N7/24/systemes_concurrents/TP3/TP-Parallel/Count java CountSequential foo 20 4
10 processeurs disponibles pour la JVM
Essai [0] : result = 5059, durée 10462 µs
Essai [1] : result = 5059, durée 8197 µs
Essai [2] : result = 5059, durée 1239 µs
Essai [3] : result = 5059, durée 800 µs
Essai [4] : result = 5059, durée 944 µs
Essai [5] : result = 5059, durée 440 µs
Essai [6] : result = 5059, durée 357 µs
Essai [7] : result = 5059, durée 802 µs
Essai [8] : result = 5059, durée 426 µs
Essai [9] : result = 5059, durée 763 µs
Essai [10] : result = 5059, durée 331 µs
Essai [11] : result = 5059, durée 726 µs
Essai [12] : result = 5059, durée 337 µs
Essai [13] : result = 5059, durée 311 µs
Essai [14] : result = 5059, durée 647 µs
Essai [15] : result = 5059, durée 302 µs
Essai [16] : result = 5059, durée 1030 µs
Essai [17] : result = 5059, durée 440 µs
Essai [18] : result = 5059, durée 280 µs
Essai [19] : result = 5059, durée 700 µs
Moyenne des durées (4 premiers essais ignorés) = 556 µs
```

FIGURE 5 – java CountSequential foo 20 20

```
Myriam@archLinux: ~/Documents/N7/24/systemes_concurrents/TP3/TP-Parallel/Count java CountThread foo 20 4
10 processeurs disponibles pour la JVM
Essai [0] : result = 5059, durée 18642 µs
Essai [1] : result = 5059, durée 2197 µs
Essai [2] : result = 5059, durée 1239 µs
Essai [3] : result = 5059, durée 800 µs
Essai [4] : result = 5059, durée 944 µs
Essai [5] : result = 5059, durée 440 µs
Essai [6] : result = 5059, durée 357 µs
Essai [7] : result = 5059, durée 802 µs
Essai [8] : result = 5059, durée 426 µs
Essai [9] : result = 5059, durée 763 µs
Essai [10] : result = 5059, durée 331 µs
Essai [11] : result = 5059, durée 726 µs
Essai [12] : result = 5059, durée 337 µs
Essai [13] : result = 5059, durée 311 µs
Essai [14] : result = 5059, durée 647 µs
Essai [15] : result = 5059, durée 302 µs
Essai [16] : result = 5059, durée 1030 µs
Essai [17] : result = 5059, durée 440 µs
Essai [18] : result = 5059, durée 280 µs
Essai [19] : result = 5059, durée 700 µs
Moyenne des durées (4 premiers essais ignorés) = 556 µs
```

FIGURE 6 – java CountThread foo 20 4

```
Myriam@archLinux: ~/Documents/N7/24/systemes_concurrents/TP3/TP-Parallel/Count java CountPool foo 20 4 8
10 processeurs disponibles pour la JVM
Essai [0] : result = 5059, durée 2399 µs
Essai [1] : result = 5059, durée 7936 µs
Essai [2] : result = 5059, durée 547 µs
Essai [3] : result = 5059, durée 138 µs
Essai [4] : result = 5059, durée 207 µs
Essai [5] : result = 5059, durée 243 µs
Essai [6] : result = 5059, durée 118 µs
Essai [7] : result = 5059, durée 131 µs
Essai [8] : result = 5059, durée 131 µs
Essai [9] : result = 5059, durée 131 µs
Essai [10] : result = 5059, durée 135 µs
Essai [11] : result = 5059, durée 113 µs
Essai [12] : result = 5059, durée 291 µs
Essai [13] : result = 5059, durée 103 µs
Essai [14] : result = 5059, durée 120 µs
Essai [15] : result = 5059, durée 205 µs
Essai [16] : result = 5059, durée 86 µs
Essai [17] : result = 5059, durée 80 µs
Essai [18] : result = 5059, durée 101 µs
Essai [19] : result = 5059, durée 135 µs
Moyenne des durées (4 premiers essais ignorés) = 148 µs
```

FIGURE 7 – java CountPool foo 20 4 8

```
Myriam@archLinux: ~/Documents/N7/24/systemes_concurrents/TP3/TP-Parallel/Count java CountForkJoin foo 20 1000
10 processeurs disponibles pour la JVM
Essai [0] : result = 5059, durée 2039 µs
Essai [1] : result = 5059, durée 1163 µs
Essai [2] : result = 5059, durée 680 µs
Essai [3] : result = 5059, durée 233 µs
Essai [4] : result = 5059, durée 237 µs
Essai [5] : result = 5059, durée 259 µs
Essai [6] : result = 5059, durée 271 µs
Essai [7] : result = 5059, durée 210 µs
Essai [8] : result = 5059, durée 230 µs
Essai [9] : result = 5059, durée 210 µs
Essai [10] : result = 5059, durée 233 µs
Essai [11] : result = 5059, durée 196 µs
Essai [12] : result = 5059, durée 197 µs
Essai [13] : result = 5059, durée 196 µs
Essai [14] : result = 5059, durée 307 µs
Essai [15] : result = 5059, durée 399 µs
Essai [16] : result = 5059, durée 307 µs
Essai [17] : result = 5059, durée 311 µs
Essai [18] : result = 5059, durée 307 µs
Essai [19] : result = 5059, durée 397 µs
Moyenne des durées (4 premiers essais ignorés) = 278 µs
```

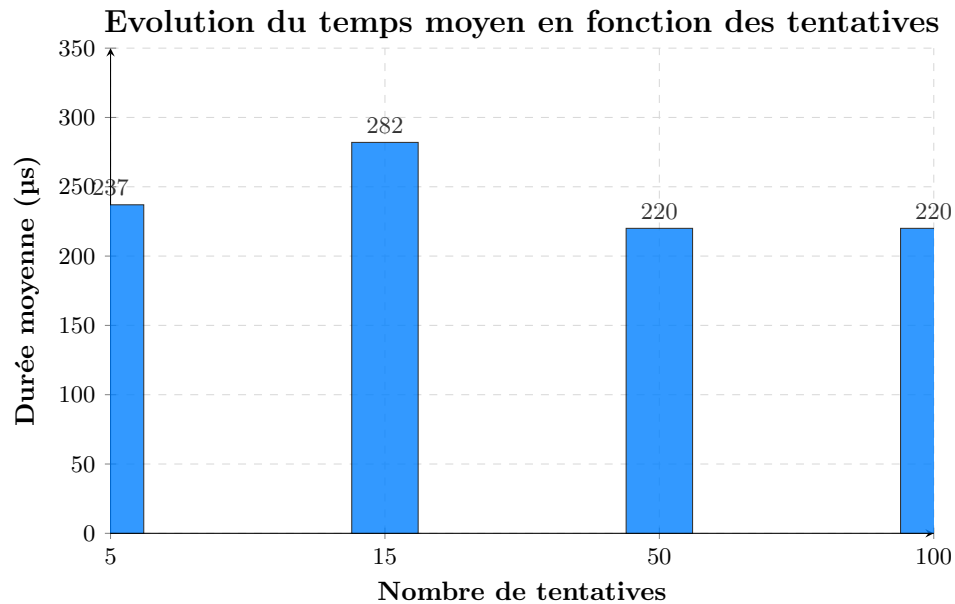
FIGURE 8 – java CountForkJoin foo 20 1000

Méthode	Temps moyen (µs)	Remarque
CountSequential	234	Exécution séquentielle simple
CountThread (4 threads)	556	Threads multiples
CountPool (4 threads, 8 tâches)	148	Pool de threads
CountForkJoin (1000 tâches)	278	Fork-Join

TABLE 2 – Comparaison des performances des différentes méthodes "Count" en fonction du temps moyen

On remarque que le résultat est bien cohérent avec le tableau créé, et que le nombre de valeur dans le tableau entre 0 et 9 est toujours de 5059. Cependant, tout comme dans la première partie, la méthode *CountPool*, avec un temps moyen de 148 µs, est la plus performante, tandis que la méthode *CountThread*, avec un temps moyen de 556 µs, est moins efficace.

2.2 Etude de la variation du nombre de threads, de la taille du pool et du seuil d'arrêt.

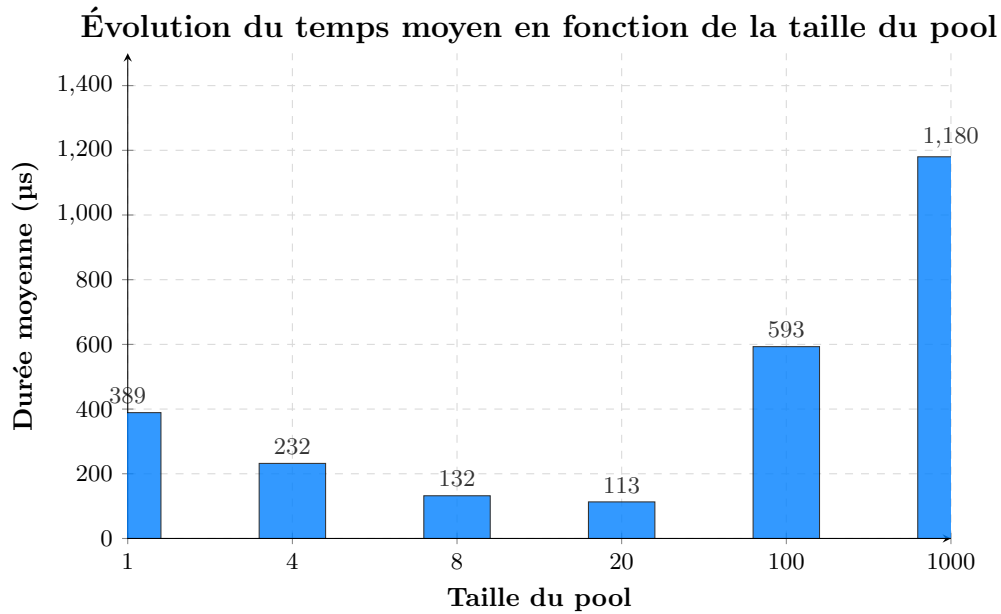


Les résultats montrent une tendance générale où la durée moyenne augmente légèrement au début, passant de 237 µs à 282 µs entre 5 et 15 tentatives. Cependant, à partir de 50 tentatives et au-delà, la durée reste stable autour de 220 µs. Cela suggère qu'après un certain nombre de tentatives, l'ajout de plus d'éléments dans le pool n'a plus d'impact significatif sur la durée d'exécution.

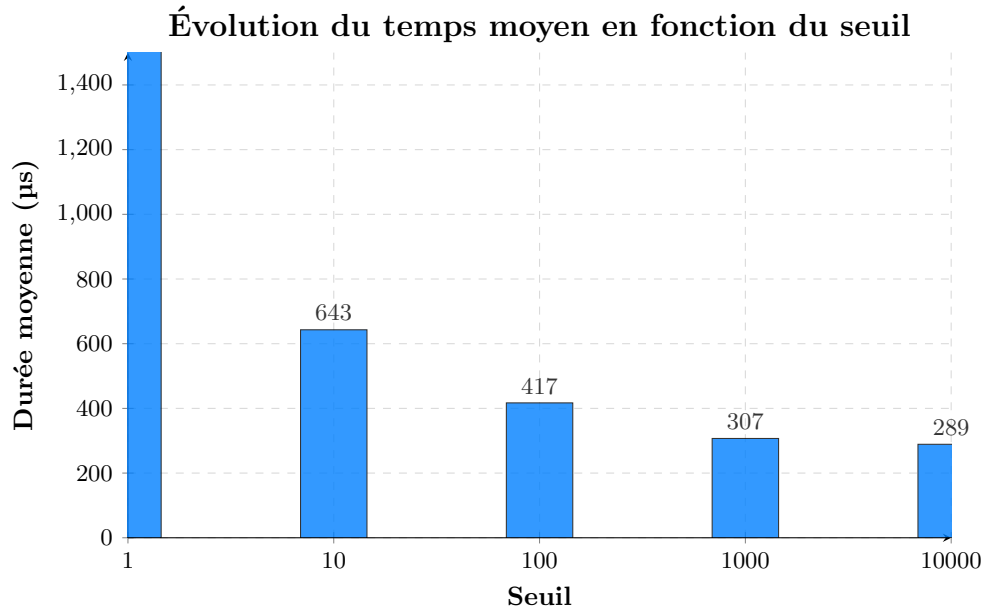
Nombre de threads	Durée moyenne (µs)
1	822
5	442
20	908
100	3663
1000	35682

TABLE 3 – Evolution du temps moyen en fonction du nombre de threads

On observe une augmentation significative du temps moyen en microsecondes avec l'augmentation du nombre de tentatives. En effet, à mesure que le nombre de tentatives passe de 1 à 1000, la durée moyenne augmente, passant de 822 µs à 35 682 µs, ce qui indique que le processus devient de plus en plus lent et coûteux en ressources avec le nombre de threads ou d'opérations simultanées.



Les résultats montrent une réduction significative des durées moyennes à mesure que le nombre de tentatives augmente, indiquant que le parallélisme des tâches améliore l'efficacité du calcul. Cependant, à partir d'une certaine taille de pool, la durée d'exécution augmente considérablement. Il existe donc une taille de pool optimale, de manière analogue à la première partie.



De manière identique à la première partie, plus on augmente le seuil et plus l'exécution est rapide.

2.3 Analyser les résultats

L'analyse des différentes méthodes pour le comptage des éléments montre que les solutions parallèles, comme l'utilisation de nombreux threads, offrent de meilleures performances que la méthode séquentielle. Toutefois, un trop grand nombre de threads peut entraîner une surcharge, réduisant les gains de performance. La solution avec un pool de threads optimisé, combinée à un seuil bien choisi, permet d'obtenir les meilleurs résultats. La méthode `CountForkJoin.java` s'est avérée la plus rapide, offrant un bon compromis entre performance et précision.

Le fichier fourni pour cette partie est : `CountSequential.java`, `CountThread.java`, `CountPool.java`, `CountForkJoin.java`.