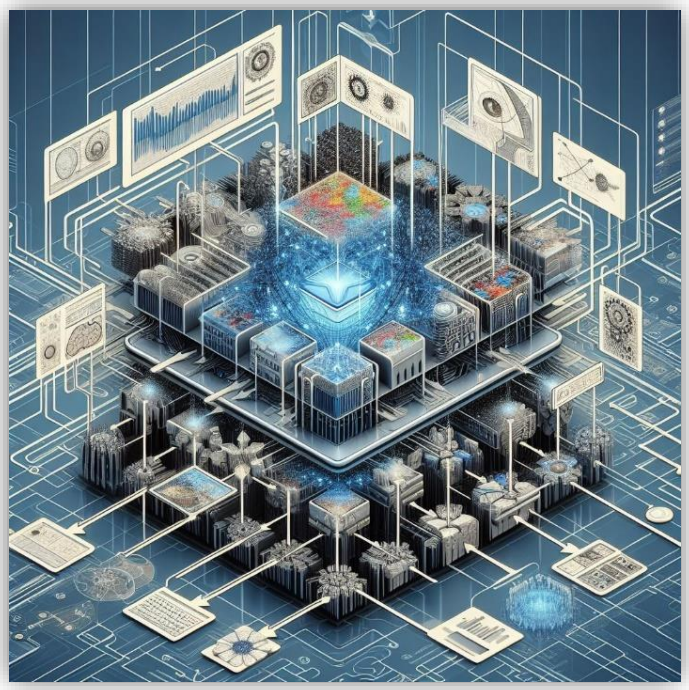


Sommaire

Spécifications de ma machine	
Processor	AMD Ryzen 7 3750H with Radeon Vega Mobile Gfx 2.30 GHz
Installed RAM	16.0 GB (13.9 GB usable)
System type	64-bit operating system, x64-based processor
Windows	Windows 11 Home

Exercice 1

Voici les résultats obtenus de l'implémentation avec des tableaux de différentes tailles et en mesurant le temps d'exécution moyen du patron PARTITION en séquentiel :

Taille des tableaux	Moyenne du temps d'exécution du patron PARTITION en séquentiel (µs)
1 024	1
16 000	17
200 000	277
4 000 000	7 930
80 000 000	156 000
200 000 000	396 000

On observe que le temps d'exécution augmente avec la taille du tableau. Cela est attendu car le nombre d'opérations à effectuer augmente avec la taille du tableau. Cependant, il est intéressant de voir que l'augmentation du temps d'exécution n'est pas linéaire. Par exemple, le temps d'exécution pour un tableau de taille 200 000 est de 277 µs, alors qu'il est de 7930 µs pour un tableau de taille 4 000 000, soit environ 20 fois plus grand. Cette augmentation non linéaire du temps d'exécution peut être dû à des facteurs comme l'efficacité de la gestion de la mémoire, le cache du processeur, la concurrence des ressources système et l'algorithme utilisé est un algorithme séquentiel donc sa complexité dépendra de la taille du tableau. Ainsi ces facteurs peuvent avoir un impact plus important sur le temps d'exécution.

Exercice 2

Voici les résultats obtenus de l'implémentation avec des tableaux de différentes tailles et en mesurant le temps d'exécution moyen du tri base utilisant le patron PARTITION en séquentiel :

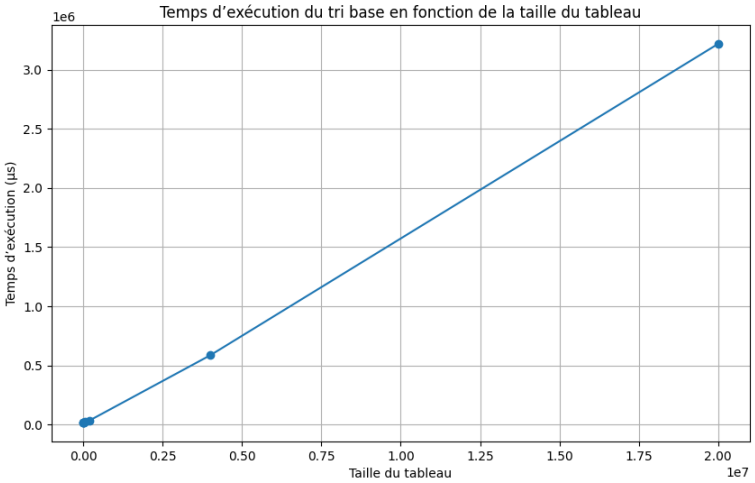
Taille des tableaux	Moyenne du temps d'exécution du tri base utilisant le patron PARTITION en séquentiel (µs)
128	39
500	110
1 024	238
4 000	2 986
16 000	10 585
200 000	73 028
4 000 000	1 604 000
20 000 000	7 336 000

On observe que le temps d'exécution augmente avec la taille du tableau. Cela est attendu car le nombre d'opérations à effectuer augmente avec la taille du tableau. Cependant, pour de petites valeurs, les performances ne sont pas optimales. Par exemple, pour un tableau de taille 128, le temps d'exécution est de 39 µs, ce qui est relativement élevé pour une si petite taille de tableau. Cela montre que l'implémentation du tri base pourrait être mieux optimisée pour gérer de petites tailles de tableau. Même raison que l'exercice précèdent les performances suboptimales pour de petites tailles de tableau peuvent être dû à des facteurs comme l'efficacité de la gestion de la mémoire, le cache du processeur, la concurrence des ressources système et l'algorithme utilisé est un algorithme séquentiel donc sa complexité dépendra de la taille du tableau. Ainsi ces facteurs peuvent avoir un impact plus important sur le temps d'exécution.

Exercice 3

Voici les résultats obtenus de l'implémentation sur des tableaux de différentes tailles et en mesurant le temps d'exécution moyen du tri base utilisant les patrons SCAN et MAP en parallèle :

Taille des tableaux	Moyenne du temps d'exécution du tri base utilisant les patrons SCAN et MAP en parallèle (µs)
128	17 279
1 024	19 254
4 000	16 965
16 000	17 911
60 000	22 539
200 000	35 261
4 000 000	586 000
20 000 000	3 218 000



On observe que pour les petits tableaux (128 à 16 000 éléments), le temps d'exécution reste relativement stable avec une légère tendance à la hausse. Cela montre que pour les petites tailles, l'overhead de la parallélisation n'affecte pas de manière significative les performances. Cependant, pour les tailles de tableau plus importantes (60 000 à 20 000 000), on observe une augmentation majeure du temps d'exécution, ce qui révèle les coûts croissants de la gestion de la parallélisation à mesure que la taille des données augmente.

L'utilisation d'un pool de threads pourrait potentiellement améliorer les performances en réduisant le temps nécessaire à la création et à la destruction des threads, permettant ainsi une gestion plus efficace des ressources. Cependant, l'overhead lié aux threads pourrait toujours être un facteur limitant, surtout si le nombre de threads dépasse le nombre de cœurs du processeur.

Exercice 4

Voici les résultats obtenus de l'implémentation sur des tableaux de différentes tailles et mesuré le temps d'exécution moyen du patron PARTITION en séquentiel (résultats du 1er exercice) et en parallèle :

Taille des tableaux	Moyenne du temps d'exécution du patron PARTITION en séquentiel (µs)	Moyenne du temps d'exécution du patron PARTITION en parallèle(µs)	Accélération	Travail (µs)	Efficacité
1 024	1	593	1 / 593 = 0.0016	8 * 593 = 4 744	1 / 4744 = 0.00021
16 000	17	668	0.025	5 344	0.0031
200 000	277	1 038	0.267	8 304	0.033
4 000 000	7 930	13 563	0.584	108 504	0.073
80 000 000	156 000	750 000	0.208	6 000 000	0.026
200 000 000	396 000	1 892 000	0.209	15 136 000	0.026

On observe que le temps d'exécution augmente avec la taille du tableau pour les deux versions, séquentielle et parallèle. Cependant, l'accélération, qui est le rapport du temps d'exécution séquentiel sur le temps d'exécution parallèle, est inférieure à 1 pour toutes les tailles de tableau. Ce qui signifie que la version parallèle est plus lente que la version séquentielle, ce qui est contraire à ce qu'on aurait pu penser. Il est possible que l'overhead de la gestion des threads dans la version parallèle soit plus important que le gain de performance dû au parallélisme, surtout pour les petites tailles de tableau. Pour les grandes tailles de tableau, l'accélération est plus grande, mais reste inférieure à 1, ce qui peut sous-entendre que la version parallèle pourrait être optimisée pour avoir des meilleures performances, en utilisant une autre version sans barrière pour effectuer l'algorithme.

En ce qui concerne le travail, on observe qu'il augmente avec la taille du tableau. Cela est attendu car plus le tableau est grand, plus il y a de données à traiter et donc plus de travail à effectuer.

Enfin, l'efficacité, est très faible pour toutes les tailles de tableau. Cela démontre que l'utilisation des ressources (c'est-à-dire les threads) n'est pas optimale. Il serait intéressant de voir d'autres technique pour l'améliorer, ainsi que la version parallèle, comme l'ajustement dynamique du nombre de threads ou l'utilisation de techniques d'équilibrage de charge.

Exercice 5

Voici les résultats obtenus de l'implémentation sur des tableaux de différentes tailles et en mesurant le temps d'exécution moyen du tri base utilisant le patron PARTITION en parallèle et le patron MAP :

Taille des tableaux	Moyenne du temps d'exécution du tri base utilisant le patron PARTITION en parallèle et le patron MAP (µs)
128	26 749
1 024	26 601
4 000	27 177
16 000	28 145
60 000	35 348
200 000	68 258
4 000 000	1 116 000
20 000 000	7 472 000

On observe que le temps d'exécution augmente avec la taille du tableau. Cela est attendu car le nombre d'opérations à effectuer augmente avec la taille du tableau. Cependant, pour de petites tailles de tableau, le temps d'exécution est relativement élevé. Par exemple, pour un tableau de taille 128, le temps d'exécution est de 26 749 µs, ce qui est significativement plus élevé que les temps d'exécution observés dans les exercices précédents.

Donc l'utilisation des patrons MAP et PARTITION en versions parallèles introduit un overhead significatif, surtout pour les petites tailles de tableau. Pour les grandes tailles de tableau, le temps d'exécution reste élevé, mais l'augmentation est moins prononcée, Donc le parallélisme apporte des bénéfices pour les grandes tailles de tableau.