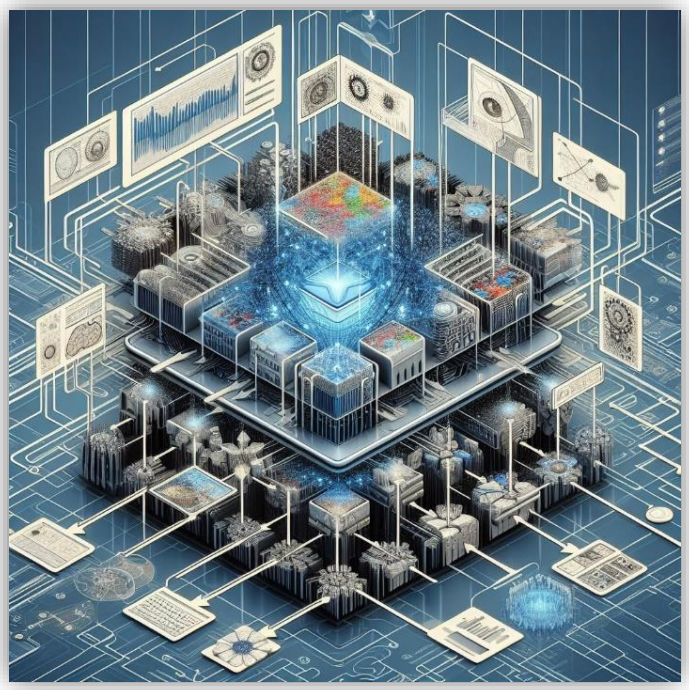

Sommaire

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

Taille des tableaux	Moyenne du temps d'exécution d'un SCAN INCLUSIF en séquentiel (µs)	Moyenne du temps d'exécution d'un SCAN INCLUSIF en parallèle (µs)	Accélération	Efficacité
1 024	3	153	3 / 153 = 0.019	3 / 153 * 8 = 0.002
16 000	37	161	0.229	0.028
200 000	444	428	1.037	0.129
4 000 000	11 156	6 046	1.845	0.230
80 000 000	214 000	124 000	1.725	0.215
200 000 000	571 000	309 000	1.848	0.230
300 000 000	855 000	490 000	1.745	0.220
500 000 000	1 406 000	751 000	1.872	0.230

On peut observer que le temps d'exécution du scan inclusif en séquentiel augmente linéairement avec la taille des tableaux, tandis que le temps d'exécution du scan inclusif en parallèle augmente plus lentement, de façon logarithmique. Cela s'explique par le fait que le nombre d'applications de la fonction en séquentiel est proportionnel à la taille des tableaux, alors qu'en parallèle, il est proportionnel au logarithme de la taille des tableaux.

La version séquentielle consiste à parcourir le tableau d'entrée du début à la fin, en appliquant l'opérateur à chaque élément et en stockant le résultat dans le tableau de sortie. Le nombre d'applications de la fonction utilisée est :

➔ $n-1$, Où n est la taille du tableau. La complexité est donc linéaire en O(n).

La version parallèle consiste à diviser le tableau d'entrée en sous-tableaux de taille égale, à assigner chaque sous-tableau à un thread, à effectuer un scan inclusif local sur chaque sous-tableau. Le nombre d'applications de la fonction utilisée est :

➔ $2(n-1) – n / p$, Où n est la taille du tableau et p est le nombre de threads. La complexité est donc logarithmique en O(log n).

On peut également observer que l'accélération du scan inclusif en parallèle par rapport au scan inclusif en séquentiel augmente avec la taille des tableaux, atteignant un maximum de 1.872 pour 500 000 000 éléments. Cela nous montre que le scan inclusif en parallèle est plus rapide que sa version en séquentiel pour les grands tableaux, mais plus lent pour les petits tableaux. Cela est expliquer par le fait que le scan inclusif en parallèle implique un coût de synchronisation et de communication entre les threads, qui devient négligeable par rapport au coût de calcul pour les grands tableaux, mais qui reste significatif pour les petits tableaux.

Donc comme pour les autres patrons parallèles vus dans le TP précédent, l'accélération devient intéressante uniquement lorsque le tableau a un grand nombre de valeurs (Plus le tableau est grand, plus l'accélération est bonne).

En ce qui concerne l'efficacité du scan inclusif en parallèle semble être assez faible, variant de 0.002 à 0.230. L'algorithme parallèle n'utilise donc pas efficacement toutes les ressources de calcul

disponibles. Cela peut être dû à plusieurs facteurs, tels que le coût de la synchronisation et de la communication entre les threads, le déséquilibre de charge entre les threads.

Exercice 2

Taille des tableaux	Moyenne du temps d'exécution d'un SCAN EXCLUSIF en séquentiel (µs)	Moyenne du temps d'exécution d'un SCAN EXCLUSIF en parallèle (µs)	Accélération	Efficacité
1 024	5	170	5 / 170 = 0.029	5 / 170 * 8 = 0.003
16 000	60	118	0.508	0.063
200 000	552	352	1.568	0.196
4 000 000	15 135	5 390	2.801	0.350
80 000 000	306 000	105 000	2.914	0.364
200 000 000	769 000	263 000	2.923	0.365
300 000 000	1 164 000	414 000	2.811	0.351
500 000 000	1 819 000	659 000	2.760	0.345

On peut observer que le temps d'exécution du scan exclusif en séquentiel augmente linéairement avec la taille des tableaux, tandis que le temps d'exécution du scan exclusif en parallèle augmente plus lentement. Cela s'explique par le fait que le nombre d'applications de la fonction en séquentiel est proportionnel à la taille des tableaux, alors qu'en parallèle, il est proportionnel au logarithme de la taille des tableaux.

La version séquentielle consiste à parcourir le tableau d'entrée du début à la fin, en appliquant l'opérateur à chaque élément et en stockant le résultat dans le tableau de sortie. Le nombre d'applications de la fonction utilisée est :

➔ $n-1$, où n est la taille du tableau.

La version parallèle consiste à diviser le tableau d'entrée en sous-tableaux de taille égale, à assigner chaque sous-tableau à un thread, à effectuer un scan exclusif local sur chaque sous-tableau. Le nombre d'applications de la fonction utilisée est :

➔ $2(n-1) - n / p + p - 1$, où n est la taille du tableau et p est le nombre de threads.

On peut également observer que l'accélération du scan exclusif en parallèle par rapport au scan exclusif en séquentiel augmente avec la taille des tableaux, atteignant un maximum de 2.923 pour 200 000 000 éléments. Cela nous montre que le scan exclusif en parallèle est plus rapide que la version en séquentiel pour les grands tableaux, mais plus lent pour les petits tableaux.

En ce qui concerne l'efficacité, on peut observer qu'elle augmente également avec la taille des tableaux, atteignant un maximum de 0.365 pour 200 000 000 éléments. Ici, l'efficacité est relativement faible, même pour les grands tableaux. Donc bien que le scan exclusif en parallèle soit plus rapide que le scan exclusif en séquentiel pour les grands tableaux, il n'utiliserait pas le parallélisme aussi efficacement qu'il le pourrait. Cela peut être dû à plusieurs facteurs, comme le coût de la synchronisation et de la communication entre les threads, le déséquilibre de charge entre les threads, ou le fait que certains threads peuvent être inactifs pendant une partie de l'exécution.