Lab 1: Somme de la suite Fibonacci en parallèle

CEG4536 - Architecture des ordinateurs III

Automne 2020

École de science informatique et de génie électrique

University of Ottawa

Professeur: Dr. Mohamed Ali Ibrahim

Miléna Dionne - 8916596

Date d'expérimentation: 18 Septembre, 2020

Date de soumission: 4 Octobre, 2020

Objectif:

Le but du laboratoire est la conception et la mise en œuvre d'une application permettant la somme de la suite de Fibonacci en utilisant la programmation parallèle.

Théorie:

La programmation parallèle est caractérisé par un système dans lequel plusieurs tâches sont logiquement actives en même temps à l'aide de threads et de coordination.

Un thread est souvent le terme utilisé dans les multiprocesseurs à mémoire partagée, tels que les microprocesseurs multi-core.

En ce qui concerne la coordination, les travaux effectués en parallèle par des threads exécutés sur différents processeurs doivent être coordonnés afin que le résultat final soit le même que si le travail a été effectué par un seul processeur. La coordination implique deux actions:

• l'une pour synchroniser des threads parallèles entre elles et

• l'autre pour communiquer les résultats partiels entre les threads, de manière à ce qu'il apparaisse comme si le travail entier était effectué par un seul processeur.

Un modèle de programmation parallèle définit comment les calculs parallèles peuvent être exprimés dans un langage de programmation de haut niveau. OpenMP est l'un des extensions linguistiques qui permet la programmation en parallèle. Ce sera également cette bibliothèque que nous utiliserons dans ce laboratoire. La plupart des constructions dans OpenMP sont des directives de compilateur.

Algorithme de solution:

Pour réalisé la série de Fibonacci en série et en parallèle, j'ai utilisé 2 concepts:

- La série Fibonacci récursive
- La série Fibonacci itérative

Dans un premier scénario, j'ai utilisé la méthode de série non-récursive (itérative) pour exécuter mon programme. Cette façon est également considérée comme la façon la plus optimale pour exécuter la série Fibonacci puisque celle-ci à un "time complexity" de O(n). En d'autres termes, il exécute la boucle dépendamment du n choisit. Cette algorithme est très facile à comprendre, à

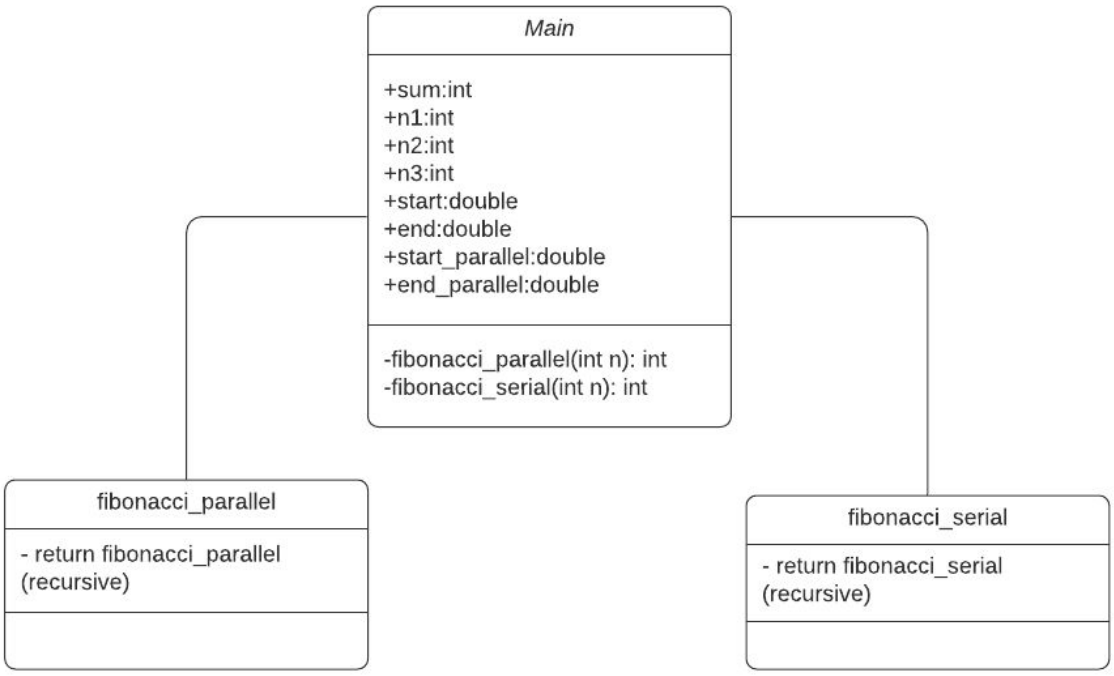
chaque itération, il additionne deux valeurs prédéfinies (n1 et n2) puis il initialise n1 par n2 et n2 par la somme trouvée.

Pour mon algorithme non-récursif en parallèle, j'ai gardé le même algorithme pour permettre de comparer les temps d'exécutions convenablement entre l'algorithme en série et celle en parallèle. Pour se faire, j'ai définit le nombre de threads que je veux exécuter pour cette algorithme puis j'ai utilisé le directive de compilation *pragma omp parallel for* pour exécuter le programme en parallèle. Puisque nous initions des valeurs différentes dans la boucle, il est important de définir chaque variables localement dans chaque threads.

Dans un deuxième scénario, j'ai utilisé la méthode de série récursive pour exécuter mon programme. Cette algorithme regarde chaque valeur (n) puis exécute une fonction dessus. Cette méthode n'est pas la plus performante puisqu'elle a un temps de complexité O(2^n) mais elle s'améliore grandement lorsque la parallélisation est utilisée. Pour cette algorithme en parallèle, j'ai ajouté le directive de compilation openMP *pragma omp parallel* avec schedule pour prendre les chunks d'itérations de chaque thread en considération jusqu'à ce que l'itération finit.

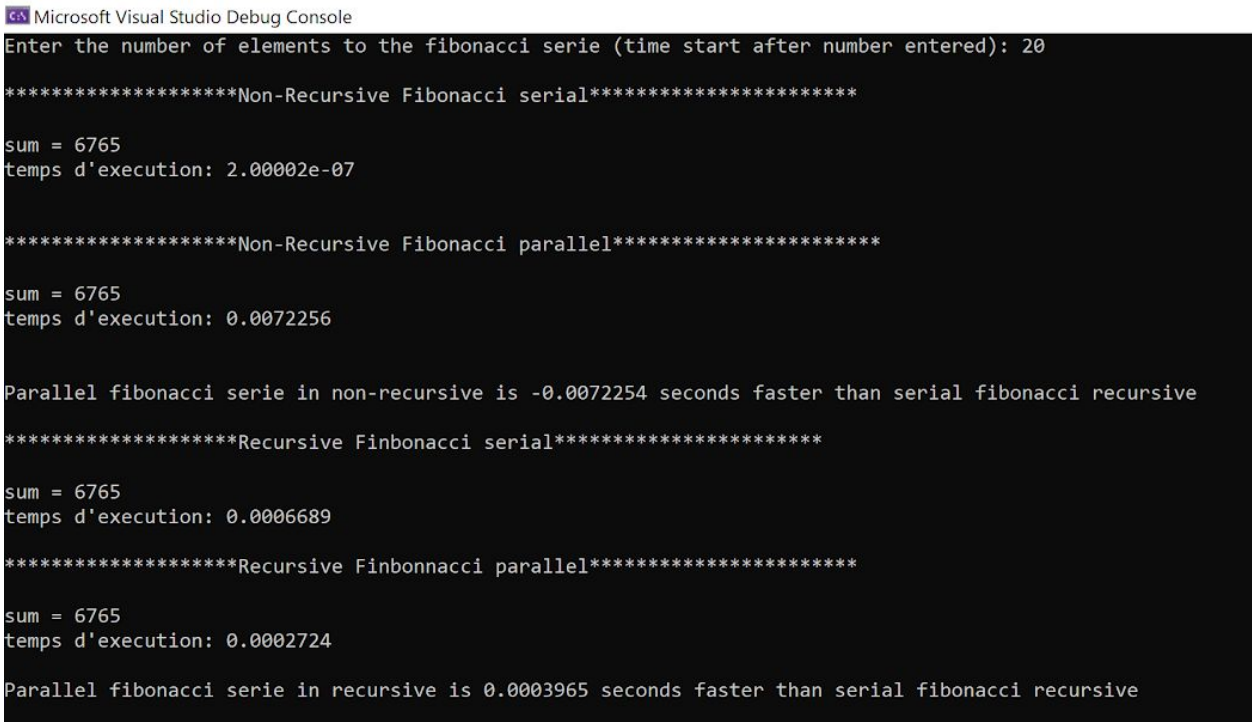
Pour chaque algorithme (série de fibonacci non-récursive en série, série de fibonacci non-récursive en parallèle, série de fibonacci récursive en série et série de fibonacci récursive en parallèle), j'ai calculé le temps d'exécution avec omp_get_wtime(). Puis j'ai calculé la différence de temps entre la série de fibonacci non-récursive en série et la série de fibonacci non-récursive en parallèle et entre la série de fibonacci récursive en série et la série de fibonacci récursive en parallèle.

Diagramme UML:

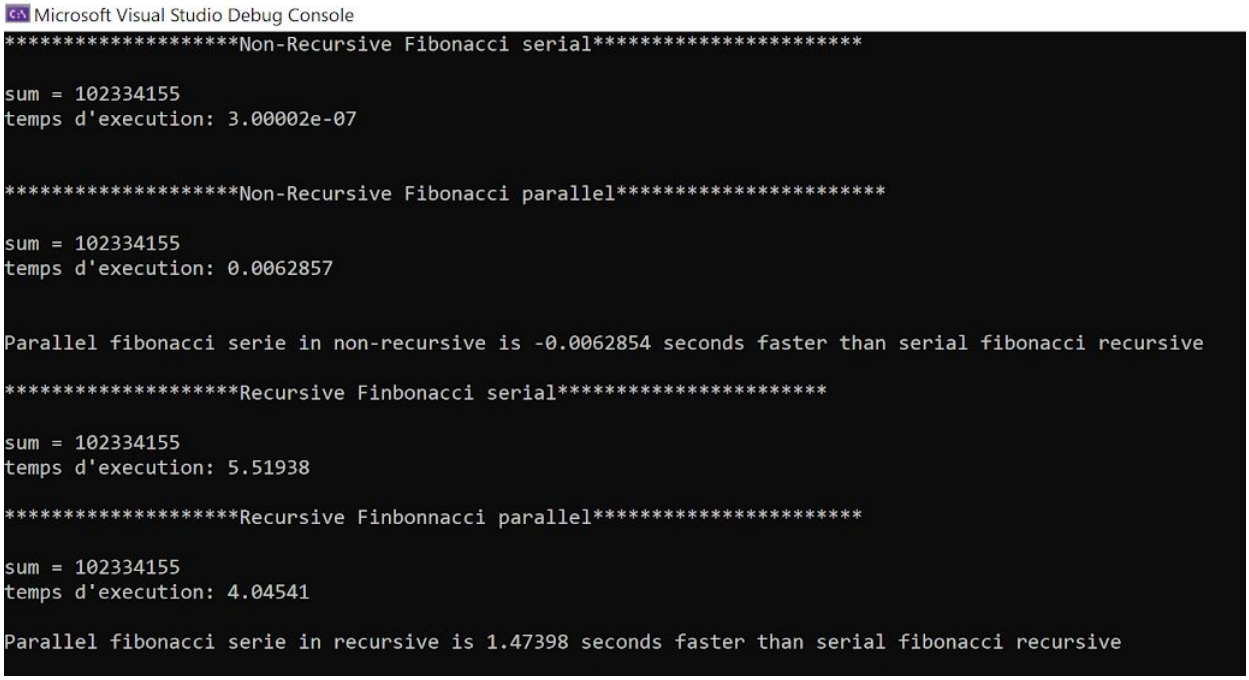


Démonstrations:

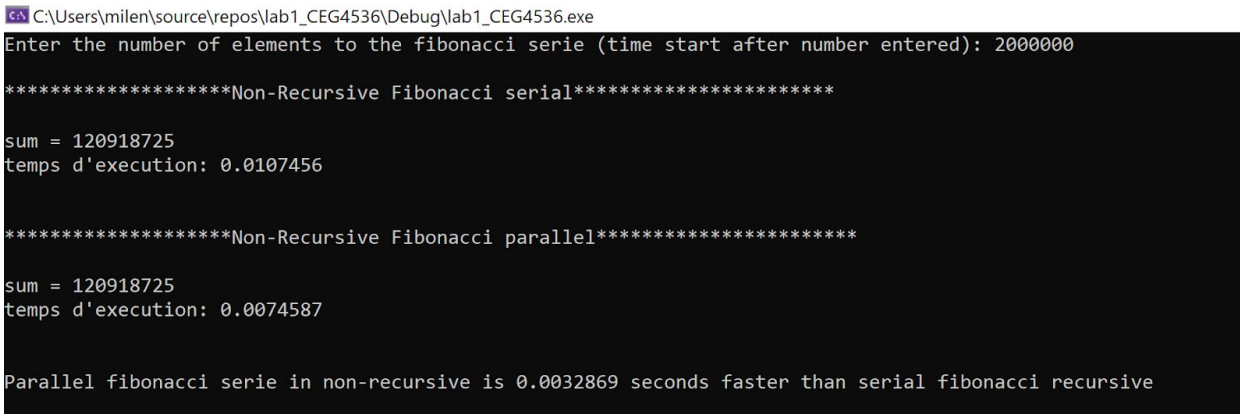
Ici est représenté la simulation de notre programme. Lorsqu'on commence l'exécution du programme, il nous est demandé d'entrer le nombre de la série de fibonacci. Lorsque le nombre est entré, le programme exécute l'algorithme fibonacci en série et en parallèle pour la récursion et les algorithmes non-récursifs. Par la suite, nous pouvons observer le temps de différence entre l'algorithme en parallèle et celle en série. Si le nombre est négatif, ceci veut dire que l'algorithme en parallèle n'était pas plus rapide que celle en série pour ce nombre donné. Le nombre de thread pour le programme a été initialisé à 16. J'ai mis les images de 3 entrées différentes soit 20, 40 et 2000000.



Capture d'écran 1: Sortie de fibonacci(20)



Capture d'écran 2: Sortie de fibonacci(40)



Capture d'écran 3: Sortie de fibonacci(2000000)

Tableau des résultats

	Non récursive fibonacci en série temps (sec)	Non récursive fibonacci en parallèle temps (sec)	Différence entre série et parallèle(sec)	Récursive fibonacci en série temps (sec)	Récursive fibonacci en parallèle temps (sec)	Différence entre série et parallèle (sec)	Résultat fibonacci
10	0.0000002	0.002	-0.002	0.000012	0.0000312	-0000192	55
20	0.0000003	0.00723	-0.00723	0.000669	0.000272	0.00040	6765
30	0.0000003	0.002202	-0.002202	0.08354	0.03601	0.047534	832040
40	0.0000003	0.006286	-0.00629	5.51938	4.04541	1.474	102334155
60	0.0000007	0.000762	-0.0007613				1820529360
200000	0.0010766	0.001027	0.00005				207797818...
1000000	0.003049	0.001915	0.0011337				188475513...
2000000	0.01075	0.007459	0.00329				120918725...

Discussion:

Le but de ce laboratoire était de concevoir et mettre en oeuvre une application permettant la somme de la suite Fibonacci en utilisant la programmation parallèle. De ce fait, le but fut atteint. J'ai pu programmer un programme en parallèle avec un nombre de threads initier à 16 et en série. Pour améliorer le résultat de mes données, j'ai programmé deux algorithmes: récursif et non-récursif.

L'algorithme récursif génère un résultat O(2^n). Cette algorithme correspond à la pire algorithme que l'on peut avoir en C++ pour la série Fibonacci. Lorsque n est 10, le résultat est de 55 et le temps d'exécution pour la série de fibonacci récursif en série est de 0.000012 secondes tandis que celle en parallèle est de 0.0000312 secondes. Ainsi, lorsque le nombre est plus petit que 10, on peut conclure que la série fibonacci est meilleur en série. Lorsque n = 20, la somme = 6765, le temps d'exécution de la série récursif en série est de 0.00075 secondes et celle en parallèle est de 0.00046 secondes. De ces données, nous pouvons conclure que l'algorithme récursif en parallèle de la série fibonacci est meilleur lorsque n est plus grand que 20 (différence de 0.00046 secondes). Lorsque nous testons avec n = 30 et n = 40, nous pouvons également remarquer que le temps est meilleur pour l'algorithme récursif en parallèle. Toutefois après n = 40, je peux remarquer le pauvre "time complexity" de l'algorithme récursif puisque celle-ci prend énormément de temps à exécuter (> 1 minute). De ce fait, il est mieux d'utiliser une autre algorithme pour des nombres plus grand comme l'algorithme non-récursif de la série fibonacci.

L'algorithme non-récursif génère un résultat O(n). Cette algorithme est considérée la plus performante pour la série Fibonacci. Cette algorithme est extrêmement rapide sur un thread (en série). Ainsi, il serait déconseillé d'utiliser la programmation en parallèle sur cette algorithme. C'est seulement lorsque n > 200000 que la série fibonacci non-récursif est meilleur en parallèle.

Conclusion:

Pour finir, la programmation en parallèle peut améliorer grandement les performances des programmes. Toutefois, plus le nombre d'itérations est petits ou que le nombre de calculs est petits, moins la programmation en parallèle est nécessaire et mois celle-ci à des chances

d’améliorer les performances. Puisque la série fibonacci est souvent utilisé pour des n petits (< 100). Il serait mieux d’utiliser la programmation en série (sur un thread).