Projet de Calcul Parallèle

Nom: Saly

Prénom : Aboubakar

Classe: M1-IABD

Exercice 1 : Parallélisme itératif

a) Performances de l'ordinateur

Caractéristique	Valeur
Modèle du processeur	AMD Ryzen 5 3500U
Nombre de cœurs-processeurs	4
Fréquence par cœur-processeur	2,10 g=GHz
RAM	17,9 Go

b) Temps d'exécution séquentiel

Taille de matrice	Temps d'exécution (s)
M=4096, N=2048, P=2048	333.418 s

c) Implémentation des programmes parallèles

Description des algorithmes utilisés (static scheduling et self scheduling)

d) Vérification de la cohérence séquentielle

Méthode de vérification et résultats

Code de comparaison des matrices

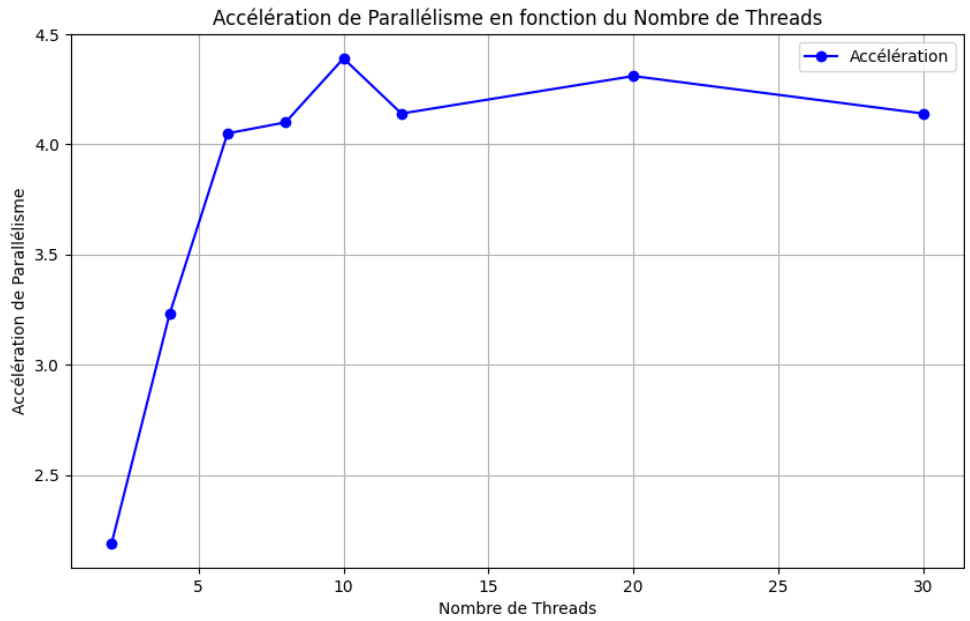
e) Évaluation des performances

Performance de MatrixProduct_parallel_static

Nombre de threads	Temps(s)	Accélération
2	152,503	2,19

4	103,332	3,23
6	82,293	4,05
8	81,262	4,10
10	76,019	4,39
12	80,514	4,14
20	77,36	4,31
30	80,622	4,14

Graphique de l'accélération de parallélisme pour MatrixProduct_parallel_static



Analyse et Interprétation

1. **Tendances générales :**
   - L'accélération de parallélisme augmente avec le nombre de threads jusqu'à environ 10 threads.
   - Après 10 threads, l'accélération se stabilise et commence même à diminuer légèrement.
2. **Observations spécifiques :**
   - **2 à 10 threads :** L'accélération augmente de manière significative. Cela montre que le programme tire profit de l'augmentation du nombre de threads pour améliorer les performances.

- **Au-delà de 10 threads :** L'accélération atteint un plateau et diminue légèrement à partir de 12 threads. Cette tendance peut être due à la surcharge de gestion des threads et aux limites de la bande passante mémoire ou du bus de communication.

3. **Interprétation :**
   - **Augmentation initiale :** L'amélioration des performances avec l'augmentation du nombre de threads est attendue, car plus de threads permettent une répartition plus efficace des tâches.
   - **Plateau et diminution :** Le plateau et la diminution de l'accélération avec plus de 10 threads suggèrent que le système atteint un point où la création et la gestion de nombreux threads ajoutent une surcharge, réduisant ainsi les gains de performance. Cela peut également indiquer une saturation des ressources disponibles (CPU, mémoire).
4. **Optimisation :**
   - Pour ce système spécifique, utiliser entre 8 et 10 threads semble être optimal pour maximiser l'accélération de parallélisme.
   - Au-delà de ce point, les gains de performance supplémentaires sont négligeables ou inexistants, et peuvent même devenir négatifs.

Conclusion

La courbe montre que l'utilisation du parallélisme peut considérablement améliorer les performances jusqu'à un certain point. Cependant, il est crucial de trouver le bon équilibre pour éviter la surcharge du système. Dans ce cas, utiliser 10 threads offre la meilleure performance avec une accélération de 4.39 fois par rapport à l'exécution séquentielle.

Performance de MatrixProduct_parallel_self

GroupSize 8

Nombre de threads	Temps(s)	Accélération
2	151.9	2.2
4	90.8	3.7
6	84.4	3.9
8	73.4	4.5

10	71.5	4.6
12	75.5	4.4
20	75.8	4.4
30	77.8	4.3

GroupSize 64

Nombre de threads	Temps(s)	Accélération
2	160.6	2.1
4	108.5	3.7
6	96.9	3.4
8	88.7	3.7
10	77.7	4.3
12	84.6	3.9
20	80.3	4.1
30	76.9	4.3

GroupSize 128

Nombre de threads	Temps(s)	Accélération
2	140.3	2.4
4	109.0	3.
6	100.1	3.3
8	89.9	3.7
10	81.4	4.1
12	82.7	4.0
20	82.5	4.

30	78.3	4.3

GroupSize 256

Nombre de threads	Temps(s)	Accélération
2	138.8	2.4
4	95.5	3.5
6	81.9	4.1
8	86.8	3.8
10	86.8	3.8
12	89.9	3.7
20	88.7	3.8
30	74.0	4.5

GroupSize 512

Nombre de threads	Temps(s)	Accélération
2	140.7	2.4
4	92.8	3.6
6	92.0	3.6
8	73.4	4.5
10	65.5	5.1
12	83.4	4.0
20	90.1	3.7
30	87.7	3.8

GroupSize 1024

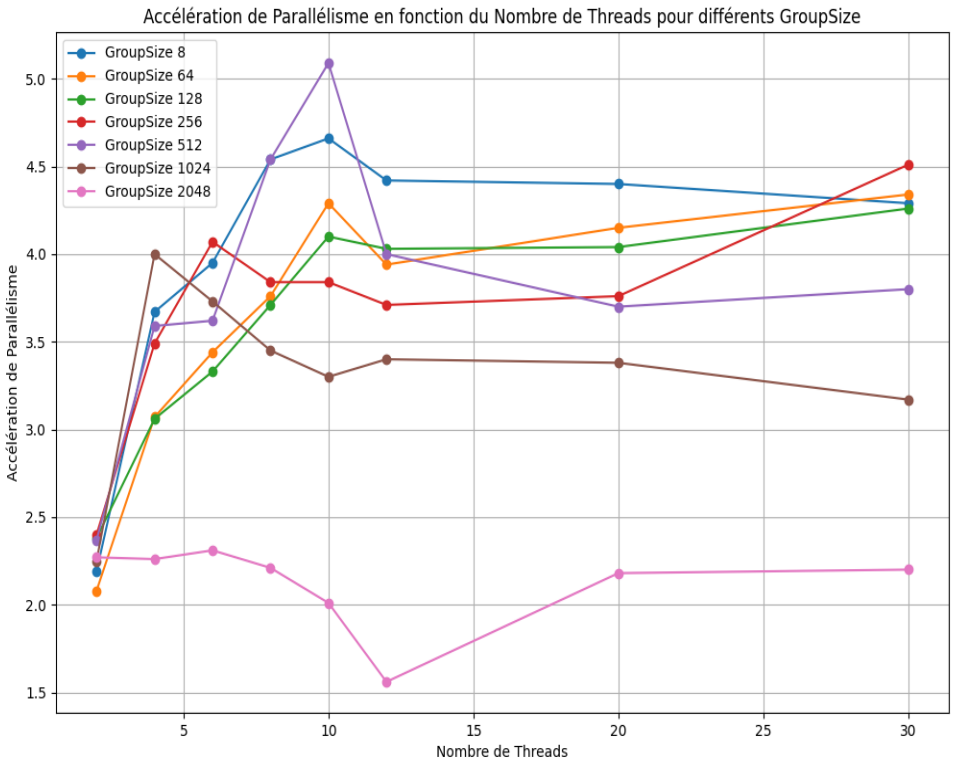
Nombre de threads	Temps(s)	Accélération
2	147.9	2.2

4	83.3	4.0
6	89.4	3.7
8	96.7	3.4
10	101.1	3.3
12	97.9	3.4
20	98.5	3.4
30	105.3	3.2

GroupSize 2048

Nombre de threads	Temps(s)	Accélération
2	146.9	2.3
4	147.3	2.3
6	144.3	2.3
8	150.6	2.2
10	165.5	2.0
12	213.6	1.5
20	152.8	2.2
30	151.7	2.2

Graphique de l'accélération de parallélisme pour MatrixProduct_parallel_self avec différentes valeurs de GroupSize



Analyse et Interprétation des Courbes

Les courbes montrent l'accélération de parallélisme en fonction du nombre de threads pour différents GroupSize. Voici une analyse détaillée des observations :

Observations Générales

Tendances Générales :

- Les courbes montrent que l'accélération augmente avec le nombre de threads jusqu'à un certain point, puis se stabilise ou diminue légèrement.
- Les GroupSize plus petits (8, 64, 128) tendent à avoir des accélérations plus élevées par rapport aux GroupSize plus grands.

GroupSize 8 :

- L'accélération augmente rapidement jusqu'à environ 10 threads, atteignant un pic à 4.66.
- Après 10 threads, l'accélération diminue légèrement et se stabilise autour de 4.29 à 30 threads.

GroupSize 64 :

- L'accélération atteint un pic de 4.34 à 30 threads.

- Il y a une augmentation régulière de l'accélération jusqu'à 10 threads, suivie d'une légère diminution et stabilisation.

GroupSize 128 :

- L'accélération augmente régulièrement jusqu'à environ 8 threads, atteignant un pic à 4.26 à 30 threads.
- L'accélération est relativement stable entre 10 et 30 threads.

GroupSize 256 :

- L'accélération atteint un pic de 4.51 à 30 threads.
- Il y a une augmentation notable jusqu'à 6 threads, suivie d'une légère diminution et stabilisation.

GroupSize 512 :

- L'accélération atteint un pic de 5.09 à 10 threads.
- Il y a une augmentation significative jusqu'à 10 threads, suivie d'une légère diminution et stabilisation.

5. **GroupSize 1024** :
   - L'accélération atteint un pic de 4.00 à 4 threads.
   - Il y a une tendance à la diminution de l'accélération après 4 threads, se stabilisant autour de 3.17 à 30 threads.

   GroupSize 2048 :
   - L'accélération reste relativement faible, avec un pic à 2.31 à 6 threads.
   - Il y a une diminution significative après 6 threads, se stabilisant autour de 2.20 à 30 threads.

Interprétation

GroupSize Optimal :

- Les GroupSize plus petits (8, 64, 128, 256) permettent une répartition plus fine du travail, réduisant la surcharge de gestion des threads et maximisant les performances.
- Les GroupSize plus grands (1024, 2048) montrent des accélérations plus faibles, ce qui suggère une saturation plus rapide des ressources du système.

Saturation et Surcharge :

- Avec des GroupSize plus grands, la surcharge de gestion des threads et la contention des ressources deviennent plus significatives, réduisant les gains de performance.
- Les GroupSize plus petits permettent de mieux tirer parti des ressources disponibles jusqu'à un certain nombre de threads avant que la surcharge ne commence à affecter les performances.

Utilisation Optimale des Threads :

- L'utilisation de 8 à 12 threads semble être optimale pour la plupart des GroupSize, maximisant l'accélération sans trop de surcharge.

- Au-delà de 12 threads, les gains de performance sont limités ou inexistants, et peuvent même devenir négatifs pour certains GroupSize.

Conclusion

L'optimisation des performances parallèles dépend de la taille des groupes (GroupSize) et du nombre de threads. Les résultats montrent qu'il est crucial de trouver un équilibre optimal pour maximiser les performances tout en évitant la surcharge du système. Les GroupSize plus petits permettent une meilleure répartition du travail, tandis que les GroupSize plus grands peuvent rapidement saturer les ressources disponibles.

En résumé, pour le système et les configurations de matrice utilisés dans cette étude, un GroupSize de 512 avec 10 threads offre la meilleure accélération de parallélisme. Cependant, l'utilisation de 8 à 12 threads semble être une règle générale pour obtenir de bonnes performances avec différents GroupSize.

Exercice 2 : Parallélisme récursif

a) Exécution du programme séquentiel

n	Temps (s)	Fibonacci(50)
50	117.639	12586269025

b) Utilisation du compilateur FJComp

Directives de compilation insérées :

```
package fjcomp;



public class Fibonacci {
    public long fibonacci(int n) {
        if (n == 0) {
            return 0;
        }
        if (n == 1) {
            return 1;
        }
        long x, y;
        //taskq nthreads=2 MaxDepth=20
        {
```

```
        //task
        x = fibonacci(n - 1);
        //task
        y = fibonacci(n - 2);
         }
        return x + y;

    }
    public static void main(String args[]){
        long startTime = System.currentTimeMillis();
          final int n=50;
          Fibonacci fib= new Fibonacci();
          long resultat=fib.fibonacci(n);
          long stopTime = System.currentTimeMillis();
          long elapsedTime = stopTime - startTime;
          System . out . println ( "Fibonacci de " + n + " est de : " +
resultat+" Temps d'exécution: "+(float)elapsedTime/1000+ " s" ) ;

    }
}
```

Code parallèle généré :

```
C:\Users\aboup\Downloads\ProjetExamenMaster_IABD_1_2023_2024bis\src\fjcomp>java -cp C:\Users\aboup\Downloads\ProjetExamenMaster_IABD_1_2023_2024bis\
src\compiler.fjcomp.jar FJCompCompiler.FJComp .\Fibonacci.fj
Lecture du fichier source...
Bravo! Parallelisation reussie.
Code parallele genere et mis en forme.
```

c) Vérification de la cohérence séquentielle

Résultat d'exécution du code parallèle généré :

```
"C:\Program Files\Java\jdk-22\bin\java.exe" "-javaagent:C:\Program
Files\JetBrains\IntelliJ IDEA 2024.1\lib\idea_rt.jar=62467:C:\Program
Files\JetBrains\IntelliJ IDEA 2024.1\bin" -Dfile.encoding=UTF-8 -
Dsun.stdout.encoding=UTF-8 -Dsun.stderr.encoding=UTF-8 -classpath
C:\Users\aboup\Downloads\ProjetExamenMaster_IABD_1_2023_2024bis\out\pro
duction\ProjetExamenMaster_IABD_1_2023_2024bis;C:\Users\aboup\Downloads
\ProjetExamenMaster_IABD_1_2023_2024bis\out\production\ProjetExamenMast
er_IABD_1_2023_2024bis\compiler.fjcomp.jar fjcomp.Fibonacci
```

Fibonacci de 50 est de : **12586269025** Temps d'exécution : **67.532 s**

Méthode de vérification et résultats : **Les résultats de Fibonacci de 50 avec les codes séquentiels et parallèles sont égaux ce qui veut dire que la cohérence séquentielle est respectée**

d) Évaluation des performances

MaxDepth 1

Nombre de threads	Temps(s)	Accélération
2	69.5	1.7
4	70.7	1.6
6	73.6	1.6
8	75.2	1.6
10	77.1	1.5
12	78.2	1.5
20	95.0	1.2
30	89.4	1.3

MaxDepth 2

Nombre de threads	Temps(s)	Accélération
2	89.5	1.3
4	56.4	2.1
6	55.1	2.1
8	45.4	2.6
10	51.5	2.3
12	48.0	2.4
20	46.8	2.5
30	46.5	2.5

MaxDepth 3

Nombre de threads	Temps(s)	Accélération
2	67.0	1.8
4	45.3	2.6
6	29.5	4.0
8	29.3	4.0
10	29.5	4.0
12	29.2	4.0
20	29.2	4.0
30	33.7	4.5

MaxDepth 4

Nombre de threads	Temps(s)	Accélération
2	77.2	1.5
4	39.8	3.0
6	29.4	4.0
8	22	5.3
10	26.1	4.5
12	26.3	4.5
20	30.5	3.8
30	29.6	3.9

MaxDepth 5

Nombre de threads	Temps(s)	Accélération
2	78.4	1.5
4	38.0	3.1

6	25.5	4.6
8	21.1	5.6
10	23.9	4.9
12	23.5	5.0
20	23.7	4.9
30	25.7	4.6

MaxDepth 6

Nombre de threads	Temps(s)	Accélération
2	74.3	1.6
4	37.3	3.1
6	24.6	4.8
8	20.5	5.7
10	21.5	5.5
12	21.1	5.6
20	21.5	5.5
30	21.1	5.6

MaxDepth 7

Nombre de threads	Temps(s)	Accélération
2	68.4	1.7
4	30.5	2.8
6	22.8	5.1
8	18.2	6.5
10	18.2	6.5
12	18.3	6.4

20	17.7	6.6
30	18.5	6.4

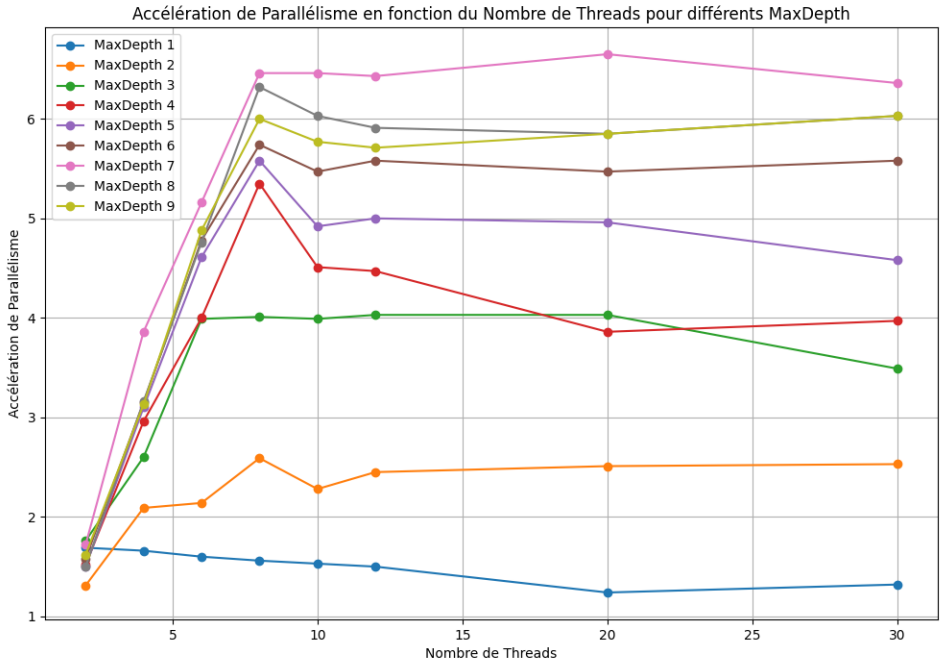
MaxDepth 8

Nombre de threads	Temps(s)	Accélération
2	78.5	1.5
4	37.2	3.2
6	24.7	4.8
8	18.6	6.3
10	19.5	6.0
12	19.9	6.0
20	20.1	6.0
30	19.5	6.0

MaxDepth 9

Nombre de threads	Temps(s)	Accélération
2	72.4	1.6
4	37.6	3.1
6	24.1	4.9
8	19.6	6.0
10	20.4	5.7
12	20.6	5.7
20	20.1	5.8
30	19.5	6.0

Graphique de l'accélération de parallélisme en fonction du Nombre de Threads pour différents MaxDepth



Analyse et Interprétation des Courbes d'Accélération de Parallélisme

Les courbes montrent l'accélération de parallélisme en fonction du nombre de threads pour différents `MaxDepth`. Voici une analyse détaillée des observations :

Observations Générales

Tendances Générales :

- Les courbes montrent que l'accélération augmente avec le nombre de threads jusqu'à un certain point, puis se stabilise ou diminue légèrement.
- Les courbes pour les valeurs de `MaxDepth` plus élevées tendent à avoir des accélérations plus élevées.

Effet du MaxDepth :

- **MaxDepth 1** : L'accélération reste assez faible et diminue légèrement avec l'augmentation du nombre de threads.
- **MaxDepth 2 à 9** : L'accélération augmente de manière significative, atteignant des pics pour certains nombres de threads avant de se stabiliser ou de diminuer.

Interprétation Spécifique

MaxDepth 1 :
- L'accélération reste autour de 1.5 à 1.7 pour tous les nombres de threads.
- Cette faible accélération peut être due à un manque de parallélisation effective car la profondeur maximale des tâches parallèles est trop faible.

MaxDepth 2 :
- L'accélération augmente rapidement pour les petits nombres de threads et se stabilise autour de 2.5 à 4 pour les threads de 6 à 30.
- Cela indique une meilleure parallélisation mais avec des limites à l'efficacité au fur et à mesure que le nombre de threads augmente.

MaxDepth 3 à 5 :
- L'accélération atteint des valeurs plus élevées, atteignant des pics pour certains nombres de threads.
- **MaxDepth 3** : Pic à 4 threads, puis diminution progressive.
- **MaxDepth 4** : Pic à 8 threads, puis diminution.
- **MaxDepth 5** : Pic à 8 threads, puis diminution.
- Cela montre une meilleure répartition des tâches et une utilisation plus efficace des threads jusqu'à un certain point.

MaxDepth 6 à 9 :
- L'accélération atteint des valeurs très élevées, particulièrement pour `MaxDepth 7` et `MaxDepth 8`.
- **MaxDepth 7** : Pic à 8 threads, puis stabilisation autour de 6.5.
- **MaxDepth 8** : Pic à 8 threads, puis légère diminution.
- **MaxDepth 9** : Pic à 8 threads, puis stabilisation autour de 6.
- Ces valeurs montrent une utilisation très efficace du parallélisme pour des valeurs élevées de `MaxDepth`.

Conclusion

6. **Optimisation du MaxDepth** :
   - Les valeurs de `MaxDepth` plus élevées (6 à 9) montrent des accélérations plus importantes, suggérant une meilleure exploitation du parallélisme.
   - Cependant, un `MaxDepth` trop élevé peut également entraîner une surcharge de gestion des threads, réduisant les gains de performance.
7. **Nombre Optimal de Threads** :
   - Pour les valeurs de `MaxDepth` plus élevées, l'utilisation de 8 à 10 threads semble être optimale, offrant une accélération maximale.
   - Au-delà de ce point, les gains supplémentaires sont limités ou inexistants, voire négatifs pour certains cas.
8. **Effet de la Saturation et de la Surcharge** :

- Les courbes montrent que la saturation des ressources et la surcharge de gestion des threads deviennent significatives au-delà de 10 threads pour les valeurs de `MaxDepth` élevées.
- Une gestion prudente des paramètres de parallélisme est nécessaire pour éviter cette surcharge.

Recommandations

9. **Pour maximiser les performances** :
    - Utiliser un `MaxDepth` de 7 ou 8.
    - Utiliser 8 à 10 threads pour maximiser l'accélération de parallélisme.
10. **Éviter la surcharge** :
    - Ne pas augmenter excessivement le nombre de threads au-delà de 10 pour les `MaxDepth` élevés.
    - Ajuster les paramètres en fonction des spécificités de la tâche et des ressources disponibles.