Systèmes Concurrents

TP CONCURRENCE ET COHERENCE

Deuxième Année, Systèmes Logiciels

Younes Saoudi

2020-2021

Efficacité de la Parallélisation

1. Le résultat « idéal » qu'on espère est que **l'ajout de de threads diminue le temps d'exécution total**
2.

Nombre de `Threads`	**Temps** `Non-Synch` **(ms)**	**Temps** `mono` **(ms)**
3	~12 200	~36 000
15	~11 200	~180 400
25	~11 000	~300 000
40	~10 800	~487 100
50	~10 800	~606 600

   - Nous nous attendons à ce que le temps d'exécution diminue avec l'ajout de threads, mais nous remarquons dans le tableau des mesures que **ce n'est pas forcément le cas**. Ceci est dû à la gestion des threads.
   - Il devient de plus en plus difficile d'évaluer le surcoût avec l'ajout de threads ; **il est assez faible** (quelques %).

Coût de la cohérence

1. Les valeurs qui seront affichées **s'il n'y avait pas de préemption** sont
   ```
   « Compteur avant : 0 »
   « Compteur avant : 0 »
   « Compteur avant : 0 »
   « Compteur après : NB_TOTAL_ITER »
   « Compteur après : NB_TOTAL_ITER »
   « Compteur après : NB_TOTAL_ITER »
   ```
2. Les valeurs qui seront affichées **si le temps processeur était partagé entre les Threads par quantum de temps** sont
   ```
   « Compteur avant : 0 »
   « Compteur après : NB_TOTAL_ITER »
   « Compteur avant : 0 »
   « Compteur après : NB_TOTAL_ITER »
   « Compteur avant : 0 »
   « Compteur après : NB_TOTAL_ITER »
   ```
3. La politique suivie par JVM pour le test est
4. **La valeur finale est différente du nombre total d'itérations** comme le montre la figure suivante :

```
Duree execution mono : 35798

Compteur avant : 0
Compteur avant : 0
Compteur avant : 0
Compteur après : 1258484450
Compteur après : 1258484450
Compteur après : 1258484450
Duree execution non synchronisee (ms): 12472
```

Ceci est dû au fait que les Threads **ne sont pas synchronisés** et interférent dans l'exécution des autres Threads.

5. **Grain Fin :**

```
Duree execution mono : 35610

Compteur avant : 0
Compteur avant : 0
Compteur avant : 0
Compteur après : 2934873004
Compteur après : 2958235182
Compteur après : 3000000000
Duree execution non synchronisee (ms): 121128
```

En **plaçant l'incrémentation dans un bloc** `synchronized`, la valeur finale du compteur atteint le nombre total d'itérations `NB_IT * NB_IT_INTERNES * nbActivites` . Le **coût de l'utilisation** de ce mécanisme est de `971%`.

6. **Grain Moyen :**

```
Compteur avant : 0
Compteur avant : 0
Compteur avant : 935637
Compteur après : 3000000000
Compteur après : 3000000000
Compteur après : 3000000000
Duree execution non synchronisee (ms): 12145
```

Le **coût de l'utilisation** de ce mécanisme **en plaçant la boucle interne dans le bloc** `synchronized` est de `-2.62%`

7. **Gros Grain :**

```
Thread t1 - compteur avant : 0
Thread t0 - compteur avant : 0
Thread t2 - compteur avant : 0
Thread t1 - compteur après : 1000000000
Thread t0 - compteur après : 2000000000
Thread t2 - compteur après : 3000000000
Duree execution non synchronisee (ms): 34990
```

Le **coût de l'utilisation** de ce mécanisme **en plaçant la boucle externe dans le bloc** `synchronized` est de `280%`.

8. En utilisant un objet de la classe `java.util.concurrent.atomic.AtomicLong` pour le compteur, nous **n'obtiendrons pas de correction** parce que rendre le compteur ainsi que l'incrémentation atomiques (avec la méthode `incrementAndGet(long)`) ne bloque pas les activités en conflit, d'où l'impossibilité **d'isoler cette opération du reste des** `Threads`.
   On obtient les résultats suivants :

```
Thread t0 - compteur avant : 0
Thread t2 - compteur avant : 0
Thread t1 - compteur avant : 0
Thread t0 - compteur après : 1790324866
Thread t2 - compteur après : 1798849117
Thread t1 - compteur après : 1800849117
Duree execution non synchronisee (ms): 112843
```

Nous remarquons ainsi que **la valeur finale du compteur n'atteint jamais** 3.10^6`L` en utilisant `AtomicLong`. De plus, le **coût de l'utilisation** de ce mécanisme est de `905%`.

9. En déclarant le compteur comme `volatile`, **nous n'arrivons pas à garantir la correction du résultat** non plus car le fait d'accéder directement à la valeur mémoire du compteur **ne garantit pas la synchronisation des** `Threads`.
   On obtient les résultats suivants :

```
Thread t0 - compteur avant : 0
Thread t2 - compteur avant : 0
Thread t1 - compteur avant : 0
Thread t0 - compteur après : 1659904617
Thread t2 - compteur après : 1676643147
Thread t1 - compteur après : 1728942229
Duree execution non synchronisee (ms): 69642
```

Nous remarquons ainsi que **la valeur finale du compteur n'atteint jamais** 3.10^6L en utilisant le mot-clé `volatile`. De plus, le **coût de l'utilisation** de ce mécanisme est de 560%.

10. Nous pouvons ainsi conclure que **le mécanisme** `synchronized` **avec grain moyen** (boucle interne dans le bloc `synchronized`) est le meilleur mécanisme puisqu'il **garantit la correction des résultats dans un temps d'exécution minimal.**