Les réponses aux questions de du TP concurrence et cohérence

Guohao DAI, Groupe L-2
Deuxième année - Département Sciences du Numérique
2021-2022

Efficacité de la parallélisation

1. Quel résultat « idéal » peut-on a priori espérer ?

Solution : Le premier type de temps de traitement de conversion inter-thread est plus long que le deuxième type de thread asynchrone. Parce que le thread asynchrone n'a pas besoin d'attendre la fin du thread précédent, mais commence directement à traiter le thread suivant.

2. Mesurer le temps d'exécution réel, en fonction de N (en faisant varier N entre 1 et 50, sans nécessairement prendre toutes les valeurs) :
   a) Expliquez les différences observées entre le temps mesuré et le temps attendu.

   Solution : Parce que lorsque nous estimons le temps d'exécution, nous n'avons pas besoin de considérer le temps d'exécution de l'instruction. En exécution réelle, l'exécution de chaque instruction prend du temps, et le temps d'exécution des processeurs aux performances différentes est également différent. Et nous avons effectué de nombreux calculs, qui vont amplifier le temps d'exécution des instructions.

   b) Evaluer le surcoût induit par la gestion des threads, au moins en principe (Il est possible que cette valeur soit trop faible pour être mesurée ainsi, les différents mécanismes d'optimisation au niveau du matériel ou du compilateur et le contexte d'exécution nuisant à la précision des mesures, cf remarque finale)

   Solution : Lorsque nous devons passer d'un thread à un autre, l'autorité prend le contrôle, effectue le ménage nécessaire (au moins enregistre et maintient la valeur), puis transfère le contrôle au thread suivant à exécuter. L'efficacité du traitement va doubler

Coût de la cohérence

1. Quelles seront a priori les valeurs affichées dans le cas où il n'y a pas préemption du processeur entre threads ?

Solution : Cela dépend du nombre de cœurs du processeur du système.

2. Quelles seront a priori les valeurs affichées dans le cas où la gestion des activités partage le temps processeur par quantum de temps entre threads ?

Solution : Le CPU exécute des tâches de façon cyclique par un algorithme d'allocation de tranche de temps. Une fois la tâche en cours exécute une tranche de temps, il passera à la tâche suivante. Cependant, l'état de la tâche précédente sera sauvegardé avant le basculement, afin que l'état de cette tâche puisse être rechargé lors du retour à cette tâche la prochaine fois.

3. Quelle est la politique effectivement suivie par la JVM utilisée pour le test ?

Solution : Dans l'ordonnancement préemptif, l'ordonnancement des threads n'a aucune autonomie, et leur temps d'exécution est également contrôlé par le CPU. Le plus gros avantage de ce mécanisme d'ordonnancement est que si un seul thread est bloqué, ce ne sera pas pour le processus. Avoir trop d'influence.

Bien sûr, bien que l'ordonnancement préemptif soit tout à fait conforme au sens du CPU, en fait, il est également possible d'améliorer les chances du thread d'obtenir les droits d'exécution du CPU en modifiant la priorité du thread.

4. La valeur finale du compteur devrait être égale au nombre total d'itérations. Vérifier que ce n'est pas le cas avec la version actuelle, et expliquer pourquoi.

Solution : Dans cette version, la valeur finale du compteur n'est pas égale au nombre total d'itérations.

5. Afin de garantir la cohérence du résultat final, on effectue les incrémentations du compteur en exclusion mutuelle, en plaçant l'incrémentation dans un bloc synchronized, associé à un objet global quelconque. (Déclarer par exemple un attribut static Object mutex = new Object(); dans la classe principale). Vérifier que le résultat est maintenant effectivement correct, et évaluer le coût de l'utilisation de ce mécanisme.
   a) en plaçant uniquement l'incrémentation de la boucle interne dans le bloc synchronized
   b) en plaçant la boucle interne dans le bloc synchronized

   (Désolé. Concernant ces deux questions, je ne sais pas comment répondre et mon programme modifié n'a pas fonctionné correctement.)

6. La correction du résultat est-elle garantie a priori si l'on utilise un objet de la classe java.util.concurrent.atomic.AtomicLong pour le compteur ? Argumenter, puis vérifier cet a priori. Evaluer le coût de l'utilisation de ce mécanisme

Solution : Le type AtomicLong est utilisé pour traiter des données entières longues. Son implémentation interne n'est pas simplement l'utilisation de synchronisation, mais un moyen plus efficace de CAS (comparer et échanger) + méthodes volatiles et natives, évitant ainsi le surcoût élevé de la synchronisation et améliorant considérablement l'efficacité d'exécution favorise.

7. La correction du résultat est-elle garantie a priori si l'on déclare le compteur comme volatile ? Argumenter, puis vérifier cet a priori. Evaluer le coût de l'utilisation de ce mécanisme.

Solution : Il existe trois concepts en programmation concurrente : l'atomicité (inséparable, avancer et reculer ensemble), la visibilité (accès multithread à une variable et tous les changements en un seul changement) et l'ordre (conformément à une certaine séquence de code pour éviter de réorganiser les instructions).

Le type volatile peut contrôler la visibilité et l'ordre, mais si l'objet variable n'est pas une opération atomique (comme i++), cela conduira à une insécurité du thread.

8. Conclure globalement sur les conditions d'utilisation (ou pas) de ces différents mécanismes.

Solution : Dans un système de multiprogrammation monoprocesseur, les processus sont exécutés en alternance, présentant ainsi une caractéristique externe d'exécution concurrente. Même si un véritable traitement parallèle ne peut pas être réalisé et que le basculement entre les processus nécessite une

certaine quantité de temps système, une exécution alternative apportera toujours de nombreux avantages en termes d'efficacité de traitement et de structure de programme.

En apparence, l'alternance et le chevauchement représentent des modes d'exécution complètement différents et des problèmes différents. En fait, ces deux technologies peuvent être considérées comme une instance de traitement concurrent, et toutes deux représentent le même problème. Dans le cas d'un seul processeur, le problème provient d'une caractéristique de base d'un système de multiprogrammation : la vitesse d'exécution relative d'un processus est imprévisible, et elle dépend des activités des autres processus, de la façon dont le système d'exploitation gère les interruptions, et de la stratégie d'ordonnancement du système d'exploitation