Série d'exercice #3

Solution

29 janvier 2019

3.1 Loi d'Amdahl

Donner la loi d'Amdahl et comment faire pour retrouver la formule.

1. À quoi cette équation s'applique.
2. Les conséquences des valeurs extrêmes valides.

3.1 Loi d'Amdahl (Solution)

Donner la loi d'Amdahl et comment faire pour retrouver la formule.

1. À quoi cette équation s'applique.
2. Les conséquences des valeurs extrêmes valides.

Solution

S : La proportion de codes séquentiels.

n : Le nombre de processeurs.

$$\text{Speedup} = \frac{1}{S + \frac{1-S}{n}}$$

1. La loi d'Amdahl établit une borne supérieur à l'accélération qu'il est possible d'obtenir en ajoutant d'autres processeurs à une application comportant des parties séquentielles et parallèles.

3.1 Loi d'Amdahl (Solution)

Donner la loi d'Amdahl et comment faire pour retrouver la formule.

1. À quoi cette équation s'applique.
2. Les conséquences des valeurs extrêmes valides.

Solution

S : La proportion de codes séquentiels.
n : Le nombre de processeurs.

$$\text{Speedup} = \frac{1}{S + \frac{1-S}{n}}$$

2. L'accélération est inversement proportionnelle à la proportion de codes séquentiels.
   L'accélération est proportionnel au nombre de processeurs.

3.2 Concurrence vs parallélisme

1. Est-il possible d'avoir de la concurrence sans parallélisme ?
2. Donnez une définition de ces deux concepts.

3.2 Concurrence vs parallélisme (Solution)

1. Est-il possible d'avoir de la concurrence sans parallélisme ?
   Oui, il suffit d'utiliser le modèle many-to-one qui lie tous les user-threads à un seul kernel-thread.
2. Donnez une définition de ces deux concepts.

3.2 Concurrence vs parallélisme (Solution)

1. Est-il possible d'avoir de la concurrence sans parallélisme ?
   Oui, il suffit d'utiliser le modèle many-to-one qui lie tous les user-threads à un seul kernel-thread.
2. Donnez une définition de ces deux concepts.

- *Concurrence* : Propriété stucturelle d'un programme selon laquel il est séparé en tâche qu'il est possible d'exécuter en parallèle.
- *Parallélisme* : Propriété d'une tâche qui s'exécute en même temps qu'une autre.

3.3 Création de *thread*

Quand un *thread* crée un nouveau *thread*, quelles parties, parmis les suivantes, sont-elles partagées entre les deux *threads* :

- Les registres
- Le tas (*heap*)
- Les variables globales
- La pile (*stack*)

3.3 Création de *thread* (Solution)

Quand un *thread* crée un nouveau *thread*, quelles parties, parmis les suivantes, sont-elles partagées entre les deux *threads* :

- Les registres
- Le tas (*heap*)
- Les variables globales
- La pile (*stack*)

Solution

Les éléments partagés entre les *threads* sont le tas, la section text (le code) et les variables globales.
Une divergence dans le fil d'exécution ce qui implique des registres et une pile propre à chaque *thread*.

3.4 L'état d'un *thread*

Pour chaque flèche dans le diagramme ci-bas, d'écrire une condition qui causerait un fil (*thread*) de passer d'un état à l'autre. Indiquer "impossible" si cela ne peut pas se produire. S'il y a lieu, indiquer aussi quelle action d'un context switch s'y produit (*save* ou *restore*).

$$\begin{array}{ccccccc} \text{En exécution} & \overset{2}{\Rightarrow} & \text{Prêt} & \overset{4}{\Rightarrow} & \text{Bloqué} & \overset{6}{\Rightarrow} & \text{En exécution} \\ (\textit{running}) & \underset{1}{\Leftarrow} & (\textit{ready}) & \underset{3}{\Leftarrow} & (\textit{blocked}) & \underset{5}{\Leftarrow} & (\textit{running}) \end{array}$$

3.4 L'état d'un *thread* (Solution)

Pour chaque flèche dans le diagramme ci-bas, d'écrire une condition qui causerait un fil (*thread*) de passer d'un état à l'autre. Indiquer "impossible" si cela ne peut pas se produire. S'il y a lieu, indiquer aussi quelle action d'un context switch s'y produit (*save* ou *restore*).

En exécution ($\overset{2}{\Rightarrow}$ / $\overset{1}{\Leftarrow}$) Prêt ($\overset{4}{\Rightarrow}$ / $\overset{3}{\Leftarrow}$) Bloqué ($\overset{6}{\Rightarrow}$ / $\overset{5}{\Leftarrow}$) En exécution

(*running*) (*ready*) (*blocked*) (*running*)

Solution

1. Ordonnancé (*scheduled* ou *restore*) ;
2. Temps expiré ou intérruption (*save*) ;
3. Résolution d'une requête d'IO ou terminaison d'un évènement ;
4. Impossible ;
5. Demande de I/O ou attente d'un évènement ;
6. Impossible

3.5 User threads

Est-il possible d'améliorer la performance d'une application multi-threaded utilisant des user-threads en exécutant le programme sur une machine ayant plusieurs processeurs plutôt que sur une machine ayant un seul processeur. Pourquoi ?

3.5 User threads (Solution)

Est-il possible d'améliorer la performance d'une application multi-threaded utilisant des user-threads en exécutant le programme sur une machine ayant plusieurs processeurs plutôt que sur une machine ayant un seul processeur. Pourquoi ?

Définition

Un user-thread est définit par les librairies d'utilisateur (i.e. POSIX pthreads, Windows threads, Java threads) alors que les kernel-threads sont définit dans le noyau.

Solution

Non, car les user-threads ne peuvent pas être exécuté en parallèle, un seul peut-être chargé dans le noyau à la fois.

3.6 Multithreading

Considérer un programme qui ouvre un fichier, effectue des opérations qui sont *cpu-bound*, puis écrit les résultats à l'intérieur d'un fichier.
Ce programme s'exécute sur une machine ayant 2 processeurs double cœur. Discuter comment vous pourriez améliorer la performance avec du multithreading.

3.6 Multithreading (Solution)

Considérer un programme qui ouvre un fichier, effectue des opérations qui sont *cpu-bound*, puis écrit les résultats à l'intérieur d'un fichier.
Ce programme s'exécute sur une machine ayant 2 processeurs double cœur. Discuter comment vous pourriez améliorer la performance avec du multithreading.

Solution

Comme la lecture et l'écriture utilise des appels systèmes bloquant (`sys_read` et `sys_write`), un unique thread peut-être utilisé pour ces deux tâches. Pour les traitements CPU-Bounds, 4 threads maximisent l'utilisation du CPU.

3.7 Fork & friends

Soit le code suivant.

```
pid_t pid1 = fork();
if (pid1 == 0)
{
  pid_t pid2 = fork();
  thread_create(...);
  thread_join(...)

  if (pid2 > 0)
    wait(NULL);
}
pid_t pid3 = fork();
```

1. Combien de processus sont créés ?
2. Combien de threads sont créés ?

3.7 Fork & friends (Solution)

Soit le code suivant.

```
pid_t pid1 = fork();
if (pid1 == 0)
{
  pid_t pid2 = fork();
  thread_create(...);
  thread_join(...)

  if (pid2 > 0)
    wait(NULL);
}
pid_t pid3 = fork();
```

1. Combien de processus sont créés ? 6 processus
2. Combien de threads sont créés ? 8 threads

3.8 Ordonnancement de threads

Dans le cours on a vu que les processus sont ordonnancés afin de bien répartir le temps de calcul entre eux. Suite à l'introduction des threads, quels nouveaux facteurs et nouvelles problématiques sont à considérer pour l'ordonnancement du modèle de processus permettant des threads ? (Considérez les unités à ordonnancer, les informations partagées, l'impact sur la performance, etc.)

3.8 Ordonnancement de threads (Solution)

Dans le cours on a vu que les processus sont ordonnancés afin de bien répartir le temps de calcul entre eux. Suite à l'introduction des threads, quels nouveaux facteurs et nouvelles problématiques sont à considérer pour l'ordonnancement du modèle de processus permettant des threads ? (Considérez les unités à ordonnancer, les informations partagées, l'impact sur la performance, etc.)

Solution

Les nouveaux facteurs à considérer dans l'ordonnancement des threads :

- On ordonnance les threads ;
- On ordonnance les processus qui ordonnancent les threads.

La premier point est la plus simple au niveau des structures à supporter puisqu'il n'y a qu'un seul type de tâche à ordonnancer. Il peut devenir difficile d'ajouter des restrictions au niveau du processus.

3.8 Ordonnancement de threads (Solution)

Dans le cours on a vu que les processus sont ordonnancés afin de bien répartir le temps de calcul entre eux. Suite à l'introduction des threads, quels nouveaux facteurs et nouvelles problématiques sont à considérer pour l'ordonnancement du modèle de processus permettant des threads ? (Considérez les unités à ordonnancer, les informations partagées, l'impact sur la performance, etc.)

Solution

Les nouveaux facteurs à considérer dans l'ordonnancement des threads :

- On ordonnance les threads ;
- On ordonnance les processus qui ordonnancent les threads.

Le deuxième point permet davantage ce genre de restriction, mais rend l'ordonnancement beaucoup plus complexe puisqu'il doit exister deux niveaux d'ordonnancements. Une bonne heuristique permetterait de maximiser le nombre de changement de contexte entre les threads d'un même processus.

3.9 Questions complémentaires

- Pourquoi voudrait-on utiliser des threads plutôt que des processus et inversement ?

Processus > Thread

Thread > Processus

3.9 Questions complémentaires (Solution)

- Pourquoi voudrait-on utiliser des threads plutôt que des processus et inversement ?

Processus > Thread

- Robustesse : un processus qui plante n'en fait pas planter d'autre en général, ce qui n'est pas le cas pour les threads.
- Sécurité/Isolation : les processus ne partagent pas leur mémoire.

Thread > Processus

3.9 Questions complémentaires (Solution)

- Pourquoi voudrait-on utiliser des threads plutôt que des processus et inversement ?

Processus > Thread

- Robustesse : un processus qui plante n'en fait pas planter d'autre en général, ce qui n'est pas le cas pour les threads.
- Sécurité/Isolation : les processus ne partagent pas leur mémoire.

Thread > Processus

- Moins couteux lors de la création et des changements de contexte, ils partagent les ressources avec les autres threads du processus.
- Ne nécessite pas l'aide du noyau pour le partage des données : elles sont automatiquement partagées entre les threads d'un même processus.

3.9 Questions complémentaires (Solution)

- Dans quelles circonstances est-il préférable d'avoir une solution multi-threaded qu'une solution single-threaded sur un système n'ayant qu'un seul processeur.

3.9 Questions complémentaires (Solution)

- Dans quelles circonstances est-il préférable d'avoir une solution multi-threaded qu'une solution single-threaded sur un système n'ayant qu'un seul processeur.

Solution

Lorsque l'on veut améliorer le temps de réponse à l'utilisateur. Lorsqu'un thread bloque, une solution multi-threaded permet au processeur d'exécuter un autre thread pendant l'attente de celui bloqué.

3.9 Questions complémentaires (Solution)

- Utiliser l'exemple d'une fabrique de chaussures pour expliquer le *data parallelism* (parallélisme de donnée) et *task parallelism* (parallélisme de tâche). [Expliquer en même temps la différence].
  De quelle manière la concurrence apparait dans ce problème ?

3.9 Questions complémentaires (Solution)

- Utiliser l'exemple d'une fabrique de chaussures pour expliquer le *data parallelism* (parallélisme de donnée) et *task parallelism* (parallélisme de tâche). [Expliquer en même temps la différence].
  De quelle manière la concurrence apparait dans ce problème ? Gophers

Definitions

- Le parallélisme de donnée (data parallelism) est une forme de parallélisation sur plusieurs processeurs. Il focus sur la distribution des données parmis les différents noeuds.
- Le parallélisme de tâche est une forme de parallélisation d'un bout de code parmis plusieurs processeurs dans un environnement parallèle. Contrairement au parallélisme de donnée il focus sur la répartition des différente tâche.

3.9 Questions complémentaires (Solution)

- Identifier les avantages et les inconvénients de chacun des modéles de multithreading. Lesquels parmis ceux-ci permettent la concurrence. Lesquels permettent du parallélisme.

Many-to-one

3.9 Questions complémentaires (Solution)

- Identifier les avantages et les inconvénients de chacun des modéles de multithreading. Lesquels parmis ceux-ci permettent la concurrence. Lesquels permettent du parallélisme.

Many-to-one

+ Les user-threads sont crées efficacements, car il ne nécessite pas la création de thread au niveau du noyau ;
- Il n'est pas parallélisable ;
- Nécessite un ordonnanceur pour les user threads.

3.9 Questions complémentaires (Solution)

- Identifier les avantages et les inconvénients de chacun des modéles de multithreading. Lesquels parmis ceux-ci permettent la concurrence. Lesquels permettent du parallélisme.

One-to-one

3.9 Questions complémentaires (Solution)

- Identifier les avantages et les inconvénients de chacun des modéles de multithreading. Lesquels parmis ceux-ci permettent la concurrence. Lesquels permettent du parallélisme.

One-to-one

+ Structure simple ;
+ Les user-thread sont parallèle ;
- Il demande la création de kernel thread pour chaque user-thread ;
- Le nombre de thread est souvant restraint du au *overhead* ;
- Un switch entre deux user-threads entrainent un kernel-thread.

3.9 Questions complémentaires (Solution)

- Identifier les avantages et les inconvénients de chacun des modéles de multithreading. Lesquels parmis ceux-ci permettent la concurrence. Lesquels permettent du parallélisme.

Many-to-Many

3.9 Questions complémentaires (Solution)

- Identifier les avantages et les inconvénients de chacun des modéles de multithreading. Lesquels parmis ceux-ci permettent la concurrence. Lesquels permettent du parallélisme.

Many-to-Many

+ N'a pas les faiblaisse des deux autres modèles ;
- Structure complexe ;
- Nécessite un ordonnanceur des user-threads.