Solutions aux exercices du chapitre 11 : OpenMP

Automne 2017

Exercice 11.1 Effet d'un appel à `foo( 5, 3 )`.

Quel est l'effet d'un appel à `foo( 5, 3 )`?

Solution :

```
foo( 5, 3 )
i = 0: id = 0
i = 1: id = 0
i = 2: id = 0
i = 3: id = 0
i = 4: id = 0
i = 0: id = 1
i = 1: id = 1
i = 2: id = 1
i = 3: id = 1
i = 4: id = 1
i = 0: id = 2
i = 1: id = 2
i = 2: id = 2
i = 3: id = 2
i = 4: id = 2
```

Exercice 11.2 Effet d'un appel à `foo( 5, 3 )`.

Quel est l'effet d'un appel à `foo( 5, 3 )`?

Solution :

```
foo( 5, 3 )
i = 0: id = 0
i = 1: id = 0
i = 4: id = 2
i = 2: id = 1
i = 3: id = 1
```

Exercice 11.3 Utilisation de `critical`.
Est-ce que cette solution sera efficace?

Solution :

Non : Section critique ⇒ accès à un verrou *à chaque itération* ☹

Exercice 11.4 Utilisation d'`atomic`.
Est-ce que cette solution sera efficace?

Solution :

Pas vraiment : Un peu plus efficace que la version avec `critical`, car les opérations atomiques sont plus efficaces que les accès à un verrou (acquisition du verrou + libération du verrou), mais quand même plus coûteux qu'une opération arithmétique régulière... d'autant plus que ces opérations atomiques se font à chaque itération ☹

Exercice 11.5 Utilisation de `reduction`.
Est-ce que cette solution sera efficace?

Solution :

Oui : Toutes les additions (intermédiaires) se font sur une copie locale de `sum`. Ce n'est que lorsque tous les *threads* ont complété que la réduction finale des résultats intermédiaires se fait au niveau de la «vraie» variable partagée.

Donc, c'est semblable à ce que fait `PRuby.preduce`!

Exercice 11.6 Utilisation `single`.

Que se passe-t-il si on omet la clause `«#pragma omp single»`?

Solution :

Plusieurs instances de `fibo(n)` vont être lancées (une instance pour chacun des *threads*de l'équipe de *threads*) et donc plusieurs impressions vont se faire — une par *thread*.

Exercice 11.7 Parallélisation du traitement des éléments d'une liste chainée.

On veut paralléliser la boucle `for`. Quels résultats produiront chacune des séries d'annotations ci-bas.

1.
```
#pragma omp parallel for
for( Noeud* pt = tete; pt != NULL; pt = pt->suivant ) {
  foo( pt );
}
```

2.
```
#pragma omp parallel
for( Noeud* pt = tete; pt != NULL; pt = pt->suivant ) {
  #pragma omp task
  foo( pt );
  #pragma omp taskwait
}
```

3.
```
#pragma omp parallel
#pragma omp single
for( Noeud* pt = tete; pt != NULL; pt = pt->suivant ) {
  #pragma omp task
  foo( pt );
  #pragma omp taskwait
}
```

4.
```
#pragma omp parallel
#pragma omp single
for( Noeud* pt = tete; pt != NULL; pt = pt->suivant ) {
  #pragma omp task
  foo( pt );
}
#pragma omp taskwait
```

Solution :

1.
```
liste-chainee.c: In function 'main':
liste-chainee.c:45:26: erreur: invalid controlling predicate
   for( Noeud* pt = tete; pt != NULL; pt = pt->suivant ) {
                          ^
liste-chainee.c:45:48: erreur: invalid increment expression
   for( Noeud* pt = tete; pt != NULL; pt = pt->suivant ) {
```

2. Plusieurs résultats possibles, mais avec pour chaque *thread* les éléments traités et imprimés en ordre :

```
foo( 0x22eb0d0 ): valeur = 5
foo( 0x22eb0d0 ): valeur = 5
foo( 0x22eb0d0 ): valeur = 5
foo( 0x22eb0b0 ): valeur = 4
foo( 0x22eb0b0 ): valeur = 4
foo( 0x22eb090 ): valeur = 3
foo( 0x22eb0b0 ): valeur = 4
foo( 0x22eb090 ): valeur = 3
foo( 0x22eb090 ): valeur = 3
foo( 0x22eb070 ): valeur = 2
foo( 0x22eb050 ): valeur = 1
foo( 0x22eb070 ): valeur = 2
foo( 0x22eb070 ): valeur = 2
foo( 0x22eb050 ): valeur = 1
foo( 0x22eb030 ): valeur = 0
foo( 0x22eb030 ): valeur = 0
foo( 0x22eb050 ): valeur = 1
foo( 0x22eb030 ): valeur = 0
```

3. Un seul et unique résultat possible = le même qu'une exécution strictement séquentielle :

```
foo( 0x13920d0 ): valeur = 5
foo( 0x13920b0 ): valeur = 4
foo( 0x1392090 ): valeur = 3
foo( 0x1392070 ): valeur = 2
foo( 0x1392050 ): valeur = 1
foo( 0x1392030 ): valeur = 0
```

4. Plusieurs résultats possibles, dont certains *qui ne seront pas en ordre*!

```
foo( 0x192d0d0 ): valeur = 5
foo( 0x192d070 ): valeur = 2
foo( 0x192d0b0 ): valeur = 4
foo( 0x192d030 ): valeur = 0
foo( 0x192d090 ): valeur = 3
foo( 0x192d050 ): valeur = 1
```