Université de Caen Basse-Normandie
U.F.R. des Sciences
Département d'informatique

Bâtiment Sciences 3 - Campus Côte de Nacre
F-14032 Caen Cédex, FRANCE

Contrôle continu 2024-2025

Niveau	**M2**
Parcours	**Informatique**
Unité d'enseignement	**SMINFO3A - Programmation parallèle et distribuée**
Responsable	**Emmanuel Cagniot** **emmanuel.cagniot@ensicaen.fr**

1 Exercice (7 pts)

Ce devoir est à effectuer en binôme. La réponse, une archive `devoir.tar.gz` *contenant le code source, devra être postée par l'un des étudiants (un seul post) sur e-campus La deadline est fixée au vendredi* 28 *novembre* 2024, 23*h*59 *(le dépôt sera clôturé à l'issue et plus aucun devoir ne sera accepté, même par e-mail).*

L'archive `devoir.tgz` *contiendra cinq répertoires, un répertoire par exercice. Chacun proposera une architecture de TP classique c'est à dire :*

- *un sous-répertoire* `src/` *contenant les définitions (*`.cpp`*) ;*
- *un sous-répertoire* `src/include` *contenant les déclarations (*`.hpp`*) ;*
- *un script de compilation* `CMakeLists.txt`*.*

1.1 Question (1,5 pts)

Écrivez une classe *template* `Count` paramétrée par, dans l'ordre :

1. un type abstrait `T` représentant un type entier ou implicitement convertible en entier ;
2. un objet `val` de type `T` ;
3. une liste `list` (longueur variable) d'éléments de type `T` (cette liste peut être vide)

et indiquant le nombre d'occurrences consécutives de `val` dans `list` à partir de son premier élément (il y a zéro occurrence si le premier élément de `list` n'est pas `val`). La figure 1 présente différentes utilisations de cette classe tandis que la figure 2 présente la trace d'exécution correspondante.

1.2 Question (2,5 pts)

Écrivez une classe *template* `Found_N_Length_Subset` paramétrée par, dans l'ordre :

1. un type abstrait `T` représentant un type entier ou implicitement convertible en entier ;
2. un entier `n` de type `size_t` ;
3. une liste `list` (longueur variable) d'éléments de type `T` (cette liste peut être vide)

```
using namespace std;

// Count.
cout << "── [Count test - begin] ──" << endl;
cout << '\t' << "Count<int, 5>::value = " << Count<int, 5>::value << endl;
cout << '\t'
     << "Count<char, 'a', 'a'>::value = " << Count<char, 'a', 'a'>::value
     << endl;
cout << '\t' << "Count<int, 5, 5, 5, 6>::value = "
          << Count<int, 5, 5, 5, 6>::value << endl;
cout << "── [Count test - end] ──" << endl;
```

FIGURE 1 – Exercice 1 : utilisations de la classe `Count`.

```
── [Count test - begin ] ──
        Count<int, 5 >::value = 0
        Count<char, 'a', 'a'>::value = 1
        Count<int, 5, 5, 5, 6>::value = 2
── [Count test - end ] ──
```

FIGURE 2 – Exercice 1 : trace d'exécution de `Count`.

et indiquant si `list` contient une sous-liste constituée de `n` occurrences d'un même élément. La figure 3 présente différentes utilisations de cette classe tandis que la figure 4 présente la trace d'exécution correspondante.

```
using namespace std;

// Found_N_Length_Subset.
cout << "── [Found_N_Length_Subset test - begin] ──" << endl;
cout << '\t' << "Found_N_Length_Subset<int, 1, 3>::Yes = "
          << Found_N_Length_Subset<int, 1, 3>::Yes << endl;
cout << '\t' << "Found_N_Length_Subset<int, 0>::Yes = "
          << Found_N_Length_Subset<int, 0>::Yes << endl;
cout << '\t' << "Found_N_Length_Subset<char, 2, 'a', 'b', 'a'>::Yes = "
          << Found_N_Length_Subset<char, 2, 'a', 'b', 'a'>::Yes << endl;
cout << '\t'
          << "Found_N_Length_Subset<char, 2, 'a', 'b', 'a', 'a'>::Yes = "
          << Found_N_Length_Subset<char, 2, 'a', 'b', 'a', 'a'>::Yes
          << endl;
cout << "── [Found_N_Length_Subset test - end] ──" << endl;
```

FIGURE 3 – Exercice 1 : utilisations de la classe `Found_N_Length_Subset`.

1.3 Question (3 pts)

Écrivez une classe *template* `Longuest_Subset` paramétrée par, dans l'ordre :

1. un type abstrait `T` représentant un type entier ou implicitement convertible en entier ;
2. une liste `list` (longueur variable) d'éléments de type `T` (cette liste peut être vide)

et retournant la longueur de sa plus longue sous-liste constituée d'un même élément. La figure 5 présente différentes utilisations de cette classe tandis que la figure 6 présente la trace d'exécution correspondante.

```
-- [Found_N_Length_Subset test - begin ] --
        Found_N_Length_Subset<int, 1, 3>::Yes = 1
        Found_N_Length_Subset<int, 0>::Yes = 1
        Found_N_Length_Subset<char, 2, 'a', 'b', 'a'>::Yes = 0
        Found_N_Length_Subset<char, 2, 'a', 'b', 'a', 'a'>::Yes = 1
-- [Found_N_Length_Subset test - end ] --
```

FIGURE 4 – Exercice 1 : trace d'exécution de `Found_N_Length_Subset`.

```
using namespace std;

// Longest_Subset.
cout << "-- [Longest_Subset test - begin ] --" << endl;
cout << '\t'
          << "Longest_Subset<int>::value = " << Longest_Subset<int>::value
          << endl;
cout << '\t' << "Longest_Subset<char, 'a'>::value = "
          << Longest_Subset<char, 'a'>::value << endl;
cout << '\t' << "Longest_Subset<int, 0, 1, 0, 0, 1, 1, 1, 0>::value = "
          << Longest_Subset<int, 0, 1, 0, 0, 1, 1, 1, 0>::value << endl;
cout << "-- [Longest_Subset test - end ] --" << endl;
```

FIGURE 5 – Exercice 1 : utilisations de la classe `Longuest_Subset`.

```
-- [Longest_Subset test - begin ] --
        Longest_Subset<int>::value = 0
        Longest_Subset<char, 'a'>::value = 1
        Longest_Subset<int, 0, 1, 0, 0, 1, 1, 1, 0>::value = 3
-- [Longest_Subset test - end ] --
```

FIGURE 6 – Exercice 1 : trace d'exécution de `Longuest_Subset`.

2 Exercice (3 pts)

Dans le TP n°4 nous avons parallélisé un algorithme de fusion récursive en OpenMP. Reprenez la correction de ce TP (ou la votre) puis, après avoir fait disparaitre toute référence à OpenMP, parallélisez ce même algorithme à l'aide de l'une des surcharges de la fonction `parallel_invoke` de TBB. Pour les mesures de durée d'exécution vous utiliserez le module `chrono` de la bibliothèque standard C++ (voir correction du TP n°7).

3 Exercice (2 pts)

Même question que l'exercice précédent mais cette fois-ci à l'aide d'un groupe de tâches TBB.

4 Exercice (5 pts)

La régression linéaire à deux variables est une méthode statistique qui consiste à considérer un ensemble de valeurs (appelées observations ou mesures) de deux propriétés x et y et à déterminer si ces dernières sont associées par une relation du type $y = a \times x + b$. Cependant, elle ne permet pas de déterminer le degré de cette association, contrairement à un autre type de méthode appelée corrélation. Dans le cas de deux variables, c'est la corrélation de Pearson qui est généralement utilisée.

Supposons que nous disposions d'un échantillons de n couples d'observations (x_i, y_i). La corrélation de Pearson commence par calculer les quantités suivantes :

1. la moyenne des valeurs x_i :
$$moy\_x = \frac{1}{n}\sum_{i=1}^{i=n} x_i, \tag{4.1}$$
2. la moyenne des valeurs y_i :
$$moy\_y = \frac{1}{n}\sum_{i=1}^{i=n} y_i, \tag{4.2}$$
3. le total des produits :
$$tot\_xy = \sum_{i=1}^{i=n} \Big\{ \big(x_i - moy\_x\big) \times \big(y_i - moy\_y\big) \Big\}, \tag{4.3}$$
4. le total des produits :
$$tot\_xx = \sum_{i=1}^{i=n} \big(x_i - moy_x\big)^2, \tag{4.4}$$
5. le total des produits :
$$tot\_yy = \sum_{i=1}^{i=n} \big(y_i - moy_y\big)^2. \tag{4.5}$$

Une fois ces calculs effectués, nous déduisons les paramètres de notre ajustement linéaire comme :

1. la valeur de a :
$$a = \frac{tot\_xy}{tot\_xx}, \tag{4.6}$$
2. la valeur de b :
$$b = moy\_y - a \times moy\_x, \tag{4.7}$$
3. la valeur du coefficient de Pearson :
$$r = \frac{tot\_xy}{\sqrt{\big(tot\_xx \times tot\_yy\big)}}. \tag{4.8}$$

Le coefficient de PEARSON, qui appartient à l'intervalle fermé $[-1,\ +1]$, mesure la qualité de l'ajustement : plus sa valeur est proche de 1 et plus les observations x et y sont corrélées.

L'archive `pearson.tar.gz`, fournie avec cet énoncé, contient un code séquentiel `pearson.cpp` permettant de calculer la degré de corrélation de deux variables x et y fournies via un fichier de mesures (un fichier de test `data_set.txt` est également fourni).

4.1 Question

Parallélisez la fonction `calculate` à l'aide de la surcharge de la fonction `parallel_reduce` de TBB dédiée aux classes (aucun mesure de temps n'est demandée).

5 Exercice (3 pts)

Dans le TP n°7 nous avons parallélisé un algorithme de fusion SPMD à l'aide de l'une des surcharges de la fonction `parallel_for` de TBB. Reprenez la correction de ce TP (ou la votre) puis, après avoir fait disparaitre toute référence à TBB, parallélisez ce même algorithme à l'aide de tâches OPENMP (P tâches si P threads disponibles). Dans le programme principal appelez directement la fonction avec le nombre de threads disponibles pour une région parallèle (ne faites pas varier le nombre de threads).