INF7235: Laboratoire #3
Utilisation des *Threading Building Blocks* d'Intel
(`parallel_for` et `parallel_reduce`)

13 février 2017
PK-4665

Obtenir le code à compléter

Obtenez une copie des fichiers pour le labo en exécutant la commande suivante sur `japet` :

```
$ git clone http://www.labunix.uqam.ca/~tremblay/git/Histogramme.git
```

Ce que vous devez faire

On veut paralléliser une fonction qui produit un **histogramme** pour les éléments entiers d'un tableau `elems`. Plus précisément, chaque nombre du tableau `elems` est un entier compris entre 0 et `valMax` (incl.). Les bornes du tableau résultant, `histo`, sont alors comprises entre 0 et `valMax` (incl.) et pour chaque valeur `val`de l'intervalle, `histo[val]` = *nombre d'occurrences de* `val` *dans* le tableau `elems`.

Par exemple, soit les valeurs suivantes et l'appel suivant à `histogramme_seq` :

```
elems == { 10, 1, 3, 3, 3, 2, 9, 1, 1, 1, 3, 10 }

histo = histogramme_seq( elems, 12, 10 )
```

Alors la valeur de `histo` après l'appel sera la suivante :

```
histo == { 0, 4, 1, 4, 0, 0, 0, 0, 0, 1, 2 }
```

En utilisant les *Threading Building Blocks* d'Intel, complétez la mise en oeuvre de la fonction `histogramme_par`. Utilisez une approche de **parallélisme de résultat** — même si cela nécessite de parcourir plusieurs fois la matrice `elems`, comme cela est fait dans la version séquentielle.

Remarques et suggestions :

- Pour compiler et exécuter le programme `histogramme.cpp`, il suffit d'exécuter la commande «`make`» (donc sans argument).
- Parallélisez tout d'abord `histogramme_par`, et ensuite parallélisez `nb_occurrences_par`. En fait, initialement de façon temporaire, vous pourriez utiliser `nb_occurrences_seq` à la place de `nb_occurrences_par`.

Mesures de temps d'exécution

Lorsque vos procédures parallèles fonctionneront, vous pouvez alors en mesures les performances à l'aide de la commande «`make bm`». Que constatez-vous?