INF5171: Laboratoire #3
Parallélisme itératif à **granularité grossière** avec parallélisme *fork/join*, parallélisme de boucles et parallélisme de données

28 septembre 2017
13h30–15h30
PK-S1565

Le but de ce laboratoire est de vous familiariser avec l'utilisation des constructions `PRuby` pour le parallélisme itératif **à granularité grossière**, et ce tant pour le parallélisme *fork/join* (constructions `pcall` et `future`) que pour le parallélisme de boucles (constructions `peach` et `peach_index`) et le parallélisme de données (constructions `pmap` et `preduce`).

---

1 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/Inversions-bis.git
```

2 Ce que vous devez faire

Comme pour le labo précédent, vous devez **compléter** diverses versions d'une méthode, définie dans une extension de la classe `Array`, permettant de trouver le nombre d'**inversions** dans un tableau :

```
[10, 20].nb_inversions.must_equal 0
[20, 10].nb_inversions.must_equal 1
[10, 10, 30, 30].nb_inversions.must_equal 0
[10, 10, 3, 3].nb_inversions.must_equal 1
[2, 1, 3, 4, 6, 5, 7, 9, 8, 10].nb_inversions.must_equal 3
```

Vous devez définir et comparer les performances de trois (3) méthodes de calcul du nombre d'inversions — la version séquentielle vous est fournie et sert d'étalon pour le calcul des accélérations :

1. `nb_inversions_par_fj_adj` : Méthode avec parallélisme itératif à granularité grossière utilisant du **parallélisme *fork/join*** — donc qui utilise `PRuby.pcall` ou `PRuby.future` — et qui répartit les éléments entre les *threads* par **tranches d'éléments adjacents**.

2. `nb_inversions_par_boucles` : Méthode avec parallélisme itératif à granularité grossière utilisant du **parallélisme de boucles** — donc qui utilise `peach` ou `peach_index`.

   Remarque : Pour cette version, attention aux interférences possibles entre les itérations de la boucle! Suggestion pour une solution sans verrou : effectuez le travail en deux (2) passes, dont au moins une d'entre elles est parallèle.

   Remarque (bis) : En fait, l'objectif ici est de vous montrer **que le parallélisme de boucles n'est pas toujours approprié ☹**

3. `nb_inversions_par_donnees` : Méthode avec **parallélisme de données** — donc qui utilise `pmap` ou `preduce`.

Pour la mesure et la comparaison des performances, un programme de mesure des temps d'exécution vous est fourni (`nb_inversions_bm.rb`). Des cibles du `makefile` sont aussi définies :

- `make bm` : Lance l'exécution du programme de mesure des temps pour trois (3) tailles de tableaux : 6 400, 64 000 et 640 000 éléments.

  Notez que pour 640 000, le temps d'exécution est un peu long... et risque de l'être encore plus si plusieurs personnes lancent son exécution en même temps ☹ Si c'est trop long, alors mettez en commentaire les lignes du `makefile` qui génèrent et traitent ce nombre d'eléments — trois lignes à mettre en commentaire.

- Des graphes peuvent ensuite être générés avec `gnuplot` puis visualisés en ouvrant les fichiers générés avec un fureteur.
  - `make graphes_temps` : Génère des graphes des temps d'exécution.
  - `make graphes_acc` : Génère des graphes des accélérations absolues.

3 Informations additionnelles

- Comme dans le laboratoire précédent divers fichiers vous sont fournis :
  - `nb_inversions.rb` : Squelette de mise en oeuvre, qui inclut un «répartiteur» (*dispatcher*) pour appeller la bonne version, selon les paramètres spécifiés par l'intermédiaire des variables d'environnement Unix — `METHODE` et `NB_THREADS`.

    Remarques :
    - La fonction suivante vous est fournie, donc utilisez la :

```
def bornes_tranche( k, nb_threads )
  DBC.require( size % nb_threads == 0,
               "*** #{nb_threads} ne divise pas #{size}" )

  (k * size / nb_threads..(k + 1) * size / nb_threads - 1)
end
```

    - Soit `r = 10..20`. Alors : `r.begin == 10` et `r.end == 20`!
  - `nb_inversions_spec.rb` : Programme de tests. Ce programme s'exécute pour un ensemble de combinaisons de versions et de nombres de *threads*. Voir le fichier `makefile` plus bas.
  - `nb_inversions_bm.rb` : Programme de mesures **des temps d'exécution**. Ce programme aussi s'exécute pour un ensemble de combinaisons de versions et de nombres de *threads*. Même remarque que pour le fichier précédent.
  - `makefile` : Fichier pour automatiser le lancement des tests et autres tâches :

    `$ make` : Exécute le programme de tests.
    La cible par défaut est celle pour les tests de la première version parallèle, donc `tests_par_fj_adj`.

    Il est possible de tester une version spécifique avec une des cibles suivantes :
    `tests_par_fj_adj`
    `tests_par_boucles`
    `tests_par_donnees`
    Pour ce faire, il suffit de spécifier explicitement la cible lors d'un appel à `make` ou de modifier la variable `TESTS` pour changer la cible par défaut.

    `$ make tests` : Lance les tests pour l'ensemble des versions parallèles.

    `$ make bm` : Lance le programme de mesures des temps d'exécution, pour les trois versions parallèles. Si l'une de vos versions ne fonctionne pas, modifiez le fichier `nb_inversions_bm.rb` pour désactiver cette version.

    `$ make graphes_temps` : Génère des graphes des temps d'exécution.
    `$ make graphes_acc` : Génère des graphes des accélérations absolues.