INF7235 : Laboratoire #6
Produit de matrices avec processus à la `go` et canaux de communication

6 mars 2017

L'objectif de ce labo est de vous familiariser avec l'utilisation de processus communiquant par l'intermédiaire de canaux, sans mémoire partagée, dans le style du langage `go`.

Ce qui est fourni

Pour obtenir le code source (sur `japet.labunix.uqam.ca`) :

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

Un `makefile` est fourni : cible par défaut = programme de test (`produit_matrices_spec.rb`).

Ce que vous devez faire

Vous devez compléter le code Ruby, dans le fichier `produit_matrices.rb`, qui effectue le produit de deux matrices — carrées et de même taille — avec des processus communiquant par l'intermédiaire de canaux, sans mémoire partagée, dans le style du langage `go`.

Le listing dans les pages qui suivent présente le code de trois méthodes, la page 4 contenant le code de la méthode à compléter — `ProduitMatrices.run`. Plus spécifiquement, il s'agit de compléter la partie envoi/réception des messages par le processus maître.

La stratégie à utiliser pour la répartition du travail est celle de **parallélisme de résultat** : chaque processus travailleur va calculer **une tranche** de la matrice résultat, tranche composée **d'un bloc de lignes adjacentes**.

Tel qu'indiqué (`ProduitMatrice.travailleur`), chaque travailleur procède comme suit :

1. Le travailleur reçoit un «morceau» de la matrice `a` et un «morceau» de la matrice `b` ;
2. Le travailleur effectue les calculs requis pour produire sa tranche du résultat ;
3. Le travailleur retourne la tranche produite au processus maître.

La principale question est de déterminer en quoi consistent ces divers «morceaux»!

Informations additionnelles :

- Un appel pour multiplier `a` et `b` se fera comme suit (cf. fichier `produit_matrices_spec.rb`), la méthode `ProduitMatrices.run` étant celle qui va créer les canaux et les processus requis, distribuer les données, puis recevoir les résultats :

```
n = ...          # Taille des matrices.
a = ...          # Matrice de taille n X n
b = ...          # Matrice de taille n X n
nb_procs = ...   # Nombre de processus a utiliser.


# Effectue le produit de a et b en repartissant le travail
# a faire entre nb_procs travailleurs.
c = ProduitMatrices.run( a, b, nb_procs )
```

- Soit `a` une matrice `n`×`n`. Alors :
  - `a[i]` = `i`^{e} ligne de `a` ;
  - `a[i..j]` (`i` ≤ `j`) = bloc allant de la `i`^{e} ligne à la `j`^{e} ligne inclusivement.
  - On peut affecter une tranche à une autre.

  Quelques exemples avec un `Array` d'entiers — mais pareil si les éléments sont des `Array` :

```
>> a = [*1..5]
=> [1, 2, 3, 4, 5]

>> a[2..3]
=> [3, 4]

>> a[0..2] = [10, 20, 30]
=> [10, 20, 30]

>> a
=> [10, 20, 30, 4, 5]
```

```
>> a[3..3] = [99]
=> [99]

>> a
=> [10, 20, 30, 99, 5]
```

- Un `Channel` peut recevoir n'importe quel type d'objet, y compris une matrice — matrice complète ou tranche/bloc de lignes.
- **Question clé** : Si on veut que le k^e processus produise la k^e tranche (le k^e bloc de lignes) de `c`... quels éléments de `a` sont alors requis vs. quels éléments de `b`?

```
class ProduitMatrices
  def self.travailleur
    lambda do |donnees, resultats|
      # On obtient les donnees a traiter.
      a = donnees.get  # Morceau approprie de a.
      b = donnees.get  # Morceau approprie de b.

      # On alloue l'espace pour la tranche resultante.
      c = Array.new(a.size) { Array.new(b.size) }

      # On calcule la tranche appropriee du resultat.
      (0...a.size).each do |i|
        (0...b.size).each do |j|
          c[i][j] = (0...b.size).reduce(0) { |s, k| s + a[i][k] * b[k][j] }
        end
      end

      # On retourne la tranche calculee au processus maitre.
      resultats << c
    end
  end

  #
```

```
  # Methode auxiliaire pour les bornes d'une tranche.
  def self.bornes_tranche( k, n, nb_procs )
    b_inf = n  / nb_procs * k
    b_sup = b_inf + n / nb_procs - 1
    b_inf..b_sup
  end

  # Multiplie a et b en utilisant exactement nb_procs travailleurs.
  #
  # Preconditions (pour simplifier):
  #   a et b sont des matrices carres de meme taille
  #   nb_procs divise n, la taille des matrices
  def self.run( a, b, nb_procs )
    unless a.size == b.size
      fail "*** Erreur: Matrices de tailles differentes"
    end
    unless a.size % nb_procs == 0
      fail "*** Erreur: nb_procs (#{nb_procs}) ne divise pas a.size (#{a.size})
    end

    # On cree les canaux.
    donnees   = Array.new( nb_procs ) { PRuby::Channel.new }
    resultats = Array.new( nb_procs ) { PRuby::Channel.new }

    # On active les travailleurs avec les canaux appropries.
    (0...nb_procs).each do |k|
      travailleur.go( donnees[k], resultats[k] )
    end

    # On transmet les donnees aux travailleurs.
    # A COMPLETER.
    # ...

    c = Array.new(a.size) { Array.new(a.size) }
    # On recoit les tranches calculees par les travailleurs.
    # A COMPLETER.
    # ...

    # On retourne le resultat a l'appelant.
    c
  end
end
```