Extraits du fichier `nb_inversions.rb`

1 Solutions style *fork/join*

Solution avec `pcall`

```
def nb_inversions_par_fj_adj
  nb_threads = [PRuby.nb_threads, size].min

  resultats = Array.new(nb_threads)
  PRuby.pcall( 0...nb_threads,
               lambda do |k|
                 bornes = bornes_de_tranche( k, nb_threads )
                 inf = k == 0 ? bornes.first : bornes.first - 1
                 resultats[k] = nb_inversions_seq_ij( inf, bornes.last )
               end
               )

  resultats.reduce(0, :+)
end
```

Solutions avec future

```ruby
# Solution #1
def nb_inversions_par_fj_adj
  nb_threads = [PRuby.nb_threads, size].min

  futures = (0...nb_threads).map do |k|
    PRuby.future do
      bornes = bornes_de_tranche( k, nb_threads )
      # Le 0-ieme thread n'a pas de tranche qui precede!
      inf = k == 0 ? bornes.first : bornes.first - 1
      nb_inversions_seq_ij( inf, bornes.last )
    end
  end

  # On force les futures puis on additionne les resultats
  # intermediaires.
  futures.map(&:value).reduce(0, :+)
end



# Solution #2
def nb_inversions_par_fj_adj
  nb_threads = [PRuby.nb_threads, size].min

  futures = (0...nb_threads).map do |k|
    PRuby.future do
      bornes = bornes_de_tranche( k, nb_threads )
      # Le 0-ieme thread n'a pas de tranche qui precede!
      inf = k == 0 ? bornes.first : bornes.first - 1
      nb_inversions_seq_ij( inf, bornes.last )
    end
  end

  # On obtient la valeur finale en additionnant les resultats
  # intermediaires, mais sans faire deux passes.
  futures.reduce(0) { |tot, future| tot + future.value }
end
```

2 Solutions avec parallélisme de boucles

```
# Solution incorrecte!
def nb_inversions_par_boucles_sans_mutex
  nb = 0
  peach_index do |k|
    nb += 1 if k > 0 && self[k-1] > self[k]
  end

  nb
end


# Solution avec resultat correct... mais pas interessante!
def nb_inversions_par_boucles_avec_mutex
  mutex = Mutex.new
  nb = 0
  peach_index do |k|
    mutex.synchronize { nb += 1 } if k > 0 &&  self[k-1] > self[k]
  end

  nb
end


# Solution en deux passes... et avec un tableau intermediaire.
def nb_inversions_par_boucles
  resultats = Array.new(size) { 0 }
  resultats.peach_index do |k|
    resultats[k] = 1 if k > 0 && self[k-1] > self[k]
  end

  resultats.reduce(0, :+)
end
```

Bref, l'utilisation du parallélisme de boucles n'est pas une approche appropriée pour ce problème, **car les itérations ne sont pas indépendantes les unes des autres.**

3 Solutions avec parallélisme de données

```ruby
# Solution en deux passes.
def nb_inversions_par_donnees
  (0...size)
    .pmap { |k| k > 0 && self[k-1] > self[k] ? 1 : 0 }
    .preduce(0, &:+)
end
```

```ruby
# Solution en une passe, avec utilisation de final_reduce.
def nb_inversions_par_donnees
  (0...size).preduce(0, final_reduce: :+) do |nb, k|
    nb + (k > 0 && self[k-1] > self[k] ? 1 : 0)
  end
end
```