Collections Parallèles

Jean-Luc Falcone

HEPIA - 2014

Progrgammation parallèle

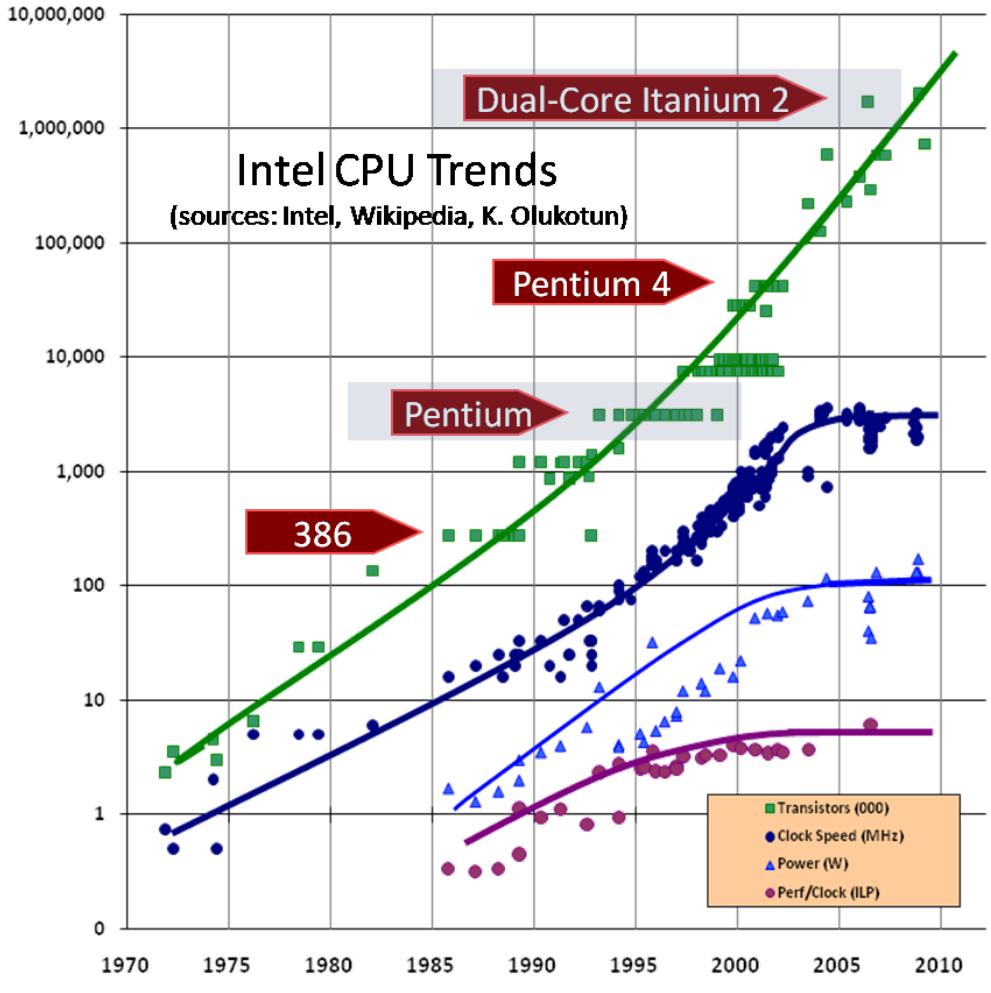
Définition: Utiliser plusieurs CPU (ou coeurs) pour:

- Exécuter plus rapidement un programme donné
- Permettre le traitement de problèmes plus volumineux.

- Mémoire partagée
- Mémoire distribuée

Big Data...

The Free Lunch Is Over !



Programmation concurrente de bas niveau

Librairies/fonctionnalités standard de Java:

- `Thread`
- `synchronized`
- `Executors`
- `Lock`
- `Atomic`
- `...`

Problèmes et difficultés

- Difficile à appréhender
- Difficile à débugger
- Problèmes habituels:
  - *Access coordination*
  - *Dead-lock*
  - *Live-lock*
  - *Starvation*
  - *Race condition*

Problème

A partir d'un texte, compter les occurences de chaque mot en ignorant les *stop words*.

Pseudo-code

```
def wordcount( text: String,
               stopWords: Set[String] ):Map[String,Int] ={

  // 1. Segmenter le texte
  // 2. Mettre en minuscules
  // 3. Enlever la ponctuation
  // 4. Enlever les stops words
  // 5. Compter les occurences

}
```

Etapes

```
private def segmentWords( text: String ): List[String] =
  text.split(" ").toList

private def removePunctuation( word: String ): String =
  word.replaceAll( "[-,.?!:;()_0-9]*", "" )

private def clean( stopWords: Set[String] ):
                                        String=>Boolean =
  w => w.nonEmpty && ! stopWords(w)
```

Solution fonctionnelle

```
def wordcount( text: String,
               stopWords: Set[String] ):Map[String,Int] =

  segmentWords( text )
    .map{ _.toLowerCase }
    .map{ removePunctuation }
    .filter{ clean(stopWords) }
    .foldLeft( Map[String,Int]() ) {
      case (m,w) =>  m + ( w -> ( m.getOrElse(w,0) + 1 ) )
    }
```

Performances

Temps en secondes pour compter les oeuvres complètes de Napoléon Bonaparte (708'247 mots).

Machine	Functional
Laptop	1.7
Desktop	1.4
Baobab	2.7

Solution procédurale

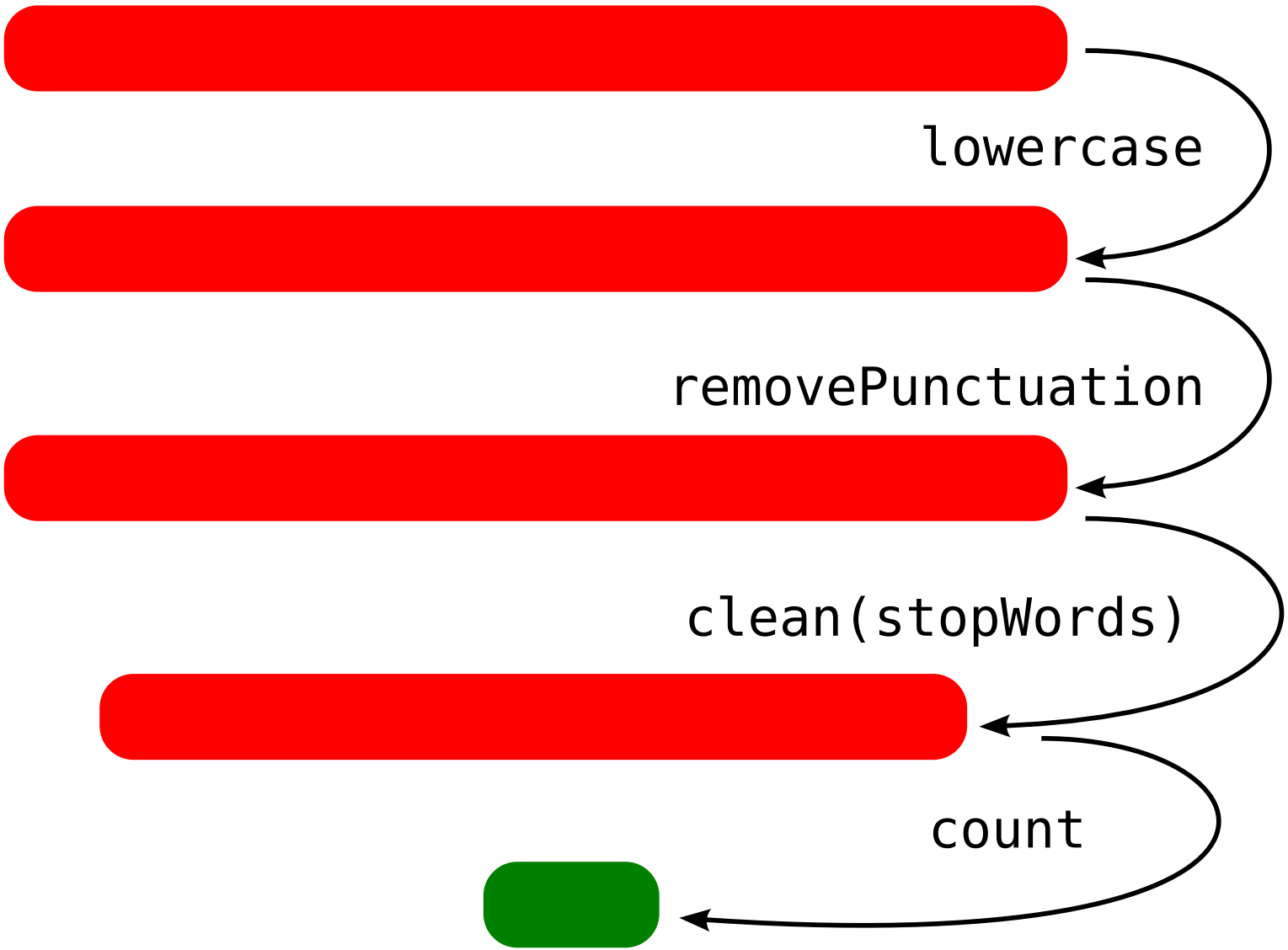
```
def wordcount( text: String, stopWords: Set[String]
                           ): HashMap[String,Int] = {
  val isClean = clean(stopWords)
  val map = new HashMap[String,Int]
  val words = segmentWords(text)
  var rest = words
  while( rest.nonEmpty ) {
    val w = rest.head
    val p = removePunctuation( w.toLowerCase )
    if( isClean( p ) ) {
      val count = if( map.contains(p) ) map(p)
                  else 0
      map += ( p -> (count+1) )
    }
    rest = rest.tail
  }
  map
```

Performances

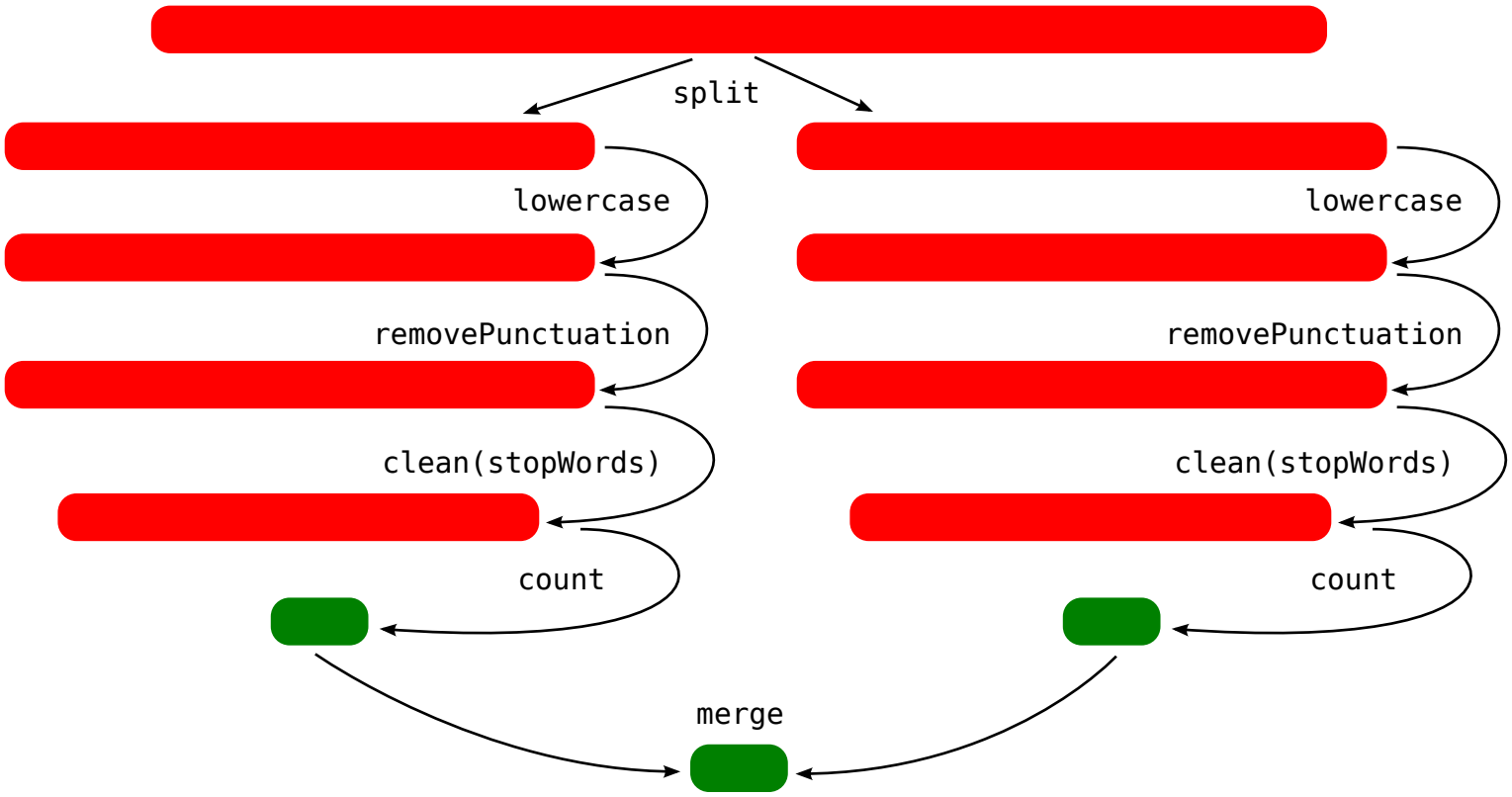
Temps en secondes pour compter les oeuvres complètes de Napoléon Bonaparte (708'247 mots).

Machine	Functional	Procedural
Laptop	1.7 (1x)	1.3 (1.3x)
Desktop	1.4 (1x)	1.1 (1.3x)
Baobab	2.7 (1x)	1.8 (1.5x)

Work-flow séquentielle



Work-flow parallèle



Solution parallèle

```
def wordcount( text: String,
               stopWords: Set[String] ): Map[String,Int] =

  segmentWords( text ).par
    .map{ _.toLowerCase }
    .map{ removePunctuation }
    .filter{ clean(stopWords) }
    .foldLeft( Map[String,Int]() ) {
      case (m,w) =>  m + ( w -> ( m.getOrElse(w,0) + 1 ) )
    }
```

Performances

Temps en secondes pour compter les oeuvres complètes de Napoléon Bonaparte (708'247 mots).

Machine	Cores	Functional	Procedural	Parallel
Laptop	2	1.7 (1x)	1.3 (1.3x)	1.3 (1.3x)
Desktop	4	1.4 (1x)	1.1 (1.3x)	0.7 (2x)
Baobab	16	2.7 (1x)	1.8 (1.5x)	0.4 (6.8x)

Collections parallèles

La méthode `.par` transforme une collection séquentielle en collection parallèle:

`mutable.Seq`	$\rightarrow$	`ParArray`
`immutable.Seq`	$\rightarrow$	`ParVector`
`Range`	$\rightarrow$	`ParRange`
`mutable.Map`	$\rightarrow$	`mutable.ParHashMap`
`mutable.Set`	$\rightarrow$	`mutable.ParHashSet`
`immutable.Map`	$\rightarrow$	`immutable.ParHashMap`
`immutable.Set`	$\rightarrow$	`immutable.ParHashSet`

Collections parallèles

La méthode `.seq` transforme une collection parallèle en collections séquentielle.

`ParArray`	$\rightarrow$	`Array`
`ParVector`	$\rightarrow$	`Vector`
`ParRange`	$\rightarrow$	`Range`
`mutable.ParHashMap`	$\rightarrow$	`mutable.HashMap`
`mutable.ParHashSet`	$\rightarrow$	`mutable.HashSet`
`immutable.ParHashMap`	$\rightarrow$	`immutable.HashMap`
`immutable.ParHashSet`	$\rightarrow$	`immutable.HashSet`

Désavantages

Attention

- Les opérations ne doivent pas causer d'effets de bord (pureté).
- Les opérations de réductions doivent être associatives.
- Les différentes opérations doivent durer le même temps pour chaque élément.
- Les méthodes `.par` et `.seq` sont coûteuses.
- Pas de contrôle fin.

Spark

La librairie Spark permet d'utiliser la même technique sur plusieurs machines (mémoire partagée):

```
val spark = new SparkContext(args(0), "SparkTest")

val myFile = spark.textFile("test.txt")
val counts = myFile.flatMap(line => line.split(" "))
                   .map(word => (word, 1))
                   .reduceByKey(_ + _)

counts.saveAsTextFile("out.txt")
```