Le calcul parallèle pour non-spécialistes, c'est maintenant!

Vincent Miele

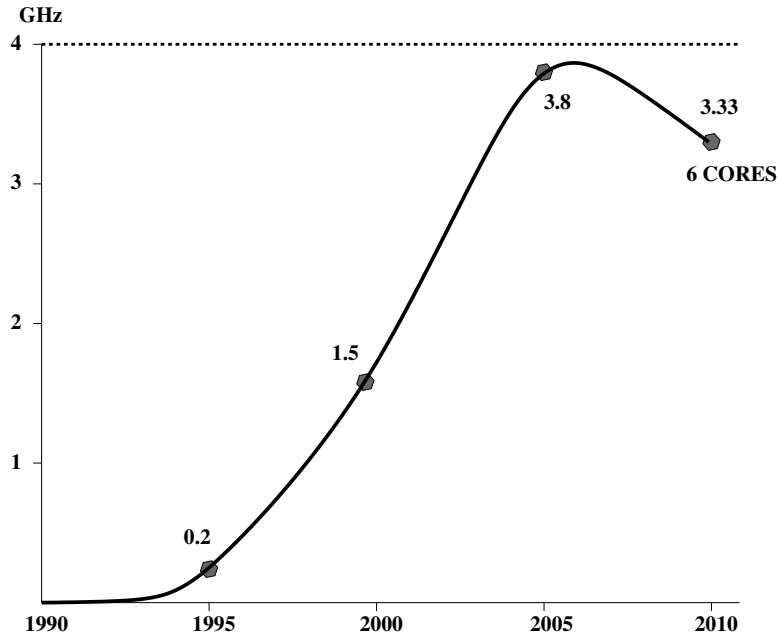
CNRS - Biométrie & Biologie Evolutive (LBBE)

Groupe Calcul (`http://calcul.math.cnrs.fr`)

16 juin 2017

« calcul parallèle », pourquoi tant de buzz ?

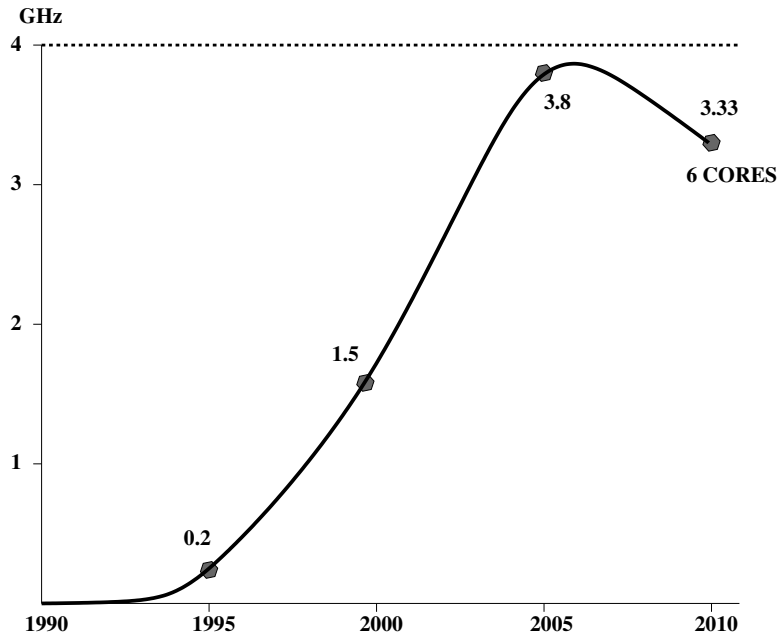
Jusque vers 2005, la performance d'une machine était directement lié au nombre d'opérations qu'elle était capable de réaliser en une seconde, c'est-à-dire à la *fréquence* de son unité de calcul (le *processeur*).



Et chaque nouvelle génération de machine voyait la fréquence de son processeur augmenter régulièrement.

« calcul parallèle », pourquoi tant de buzz ?

Mais l'augmentation de la fréquence d'un processeur entraîne une augmentation de la quantité de chaleur à dissiper
→ non rentable pour les fondeurs de refroidir des processeurs à fréquence élevé



→ les fréquences se sont donc élevées jusqu'aux alentours de 4 GHz pour finalement se stabiliser entre 3 et 4 GHz.

« calcul parallèle », pourquoi tant de buzz ?

Petite révolution : les fondeurs ont réussi à graver leurs transistors de plus en plus finement...
→ 2 puis 4, 6 ou 8 unité de calcul dans le processeur !
→ potentiellement des machines + performantes...
... potentiellement seulement (un code ne peut pas, tel quel, utiliser plusieurs unités de calcul).

« calcul parallèle », pourquoi tant de buzz ?

Petite révolution : les fondeurs ont réussi à graver leurs transistors de plus en plus finement...
→ 2 puis 4, 6 ou 8 unité de calcul dans le processeur !
→ potentiellement des machines + performantes...
... potentiellement seulement (un code ne peut pas, tel quel, utiliser plusieurs unités de calcul).

Problème de dissipation thermique : légère baisse de la fréquence des processeurs pour pouvoir accueillir un plus grand nombre d'unités de calcul...

« calcul parallèle », pourquoi tant de buzz ?

Petite révolution : les fondeurs ont réussi à graver leurs transistors de plus en plus finement...
→ 2 puis 4, 6 ou 8 unité de calcul dans le processeur !
→ potentiellement des machines + performantes...
... potentiellement seulement (un code ne peut pas, tel quel, utiliser plusieurs unités de calcul).

Problème de dissipation thermique : légère baisse de la fréquence des processeurs pour pouvoir accueillir un plus grand nombre d'unités de calcul...

« *The free lunch is over* » (Sutter, 2005) : on ne peut plus gagner en performance de façon gratuite, juste en changeant de machine...
Il faut « penser parallèle » !

[Digression] « c'est maintenant », vraiment ?

« Ne jamais optimiser prématurément » (Sutter & Alexandrescu, 2004).

Avant de paralléliser, s'organiser :

- Suivi des modifications (`git`)
- Vérification de l'exactitude du résultat → des tests (`testthat`)

  « Le problème, ce n'est pas qu'on n'a pas fait de tests, mais c'est qu'on ne les a pas gardés... et qu'on ne peut plus les refaire tourner automatiquement. » (Wickham, 2014)
- Mesurer les performances temps et mémoire (`microbenchmark` + `Rprof`)

  « Les programmeurs gâchent énormément de temps en se focalisant sur des parties de leurs codes qui au final ne sont pas critiques ! » (Knuth, 1974)

[Digression] « c'est maintenant », vraiment ?

Avant de paralléliser, optimiser les points chauds :

- Améliorer le code `R` 
- Implémenter les points chauds avec un langage compilé (`Rcpp`) 

« c'est maintenant... », vraiment ?

Oui mais après avoir épuisé ces recettes, se tourner vers l'idée de combiner la puissance de plusieurs unités de calcul pour gagner en performance.

Simple ? L'exemple du puzzle.

Fondamentaux du calcul parallèle - Vocabulaire

POINT n°0 : savoir de quoi on parle

Code séquentiel : conçu, depuis l'algorithme jusqu'à son implémentation, pour être exécuté sur une unique unité de calcul.

Calcul réparti : exécuter un code unique et séquentiel pour plusieurs jeux d'entrées. Chaque exécution est indépendante et utilise une unité de calcul.
Exple : analyse de sensibilité

Calcul parallèle : exécuter un code pour un jeu d'entrées unique avec plusieurs unités de calcul et obtenir un résultat unique.
Exple : produit matrice-vecteur

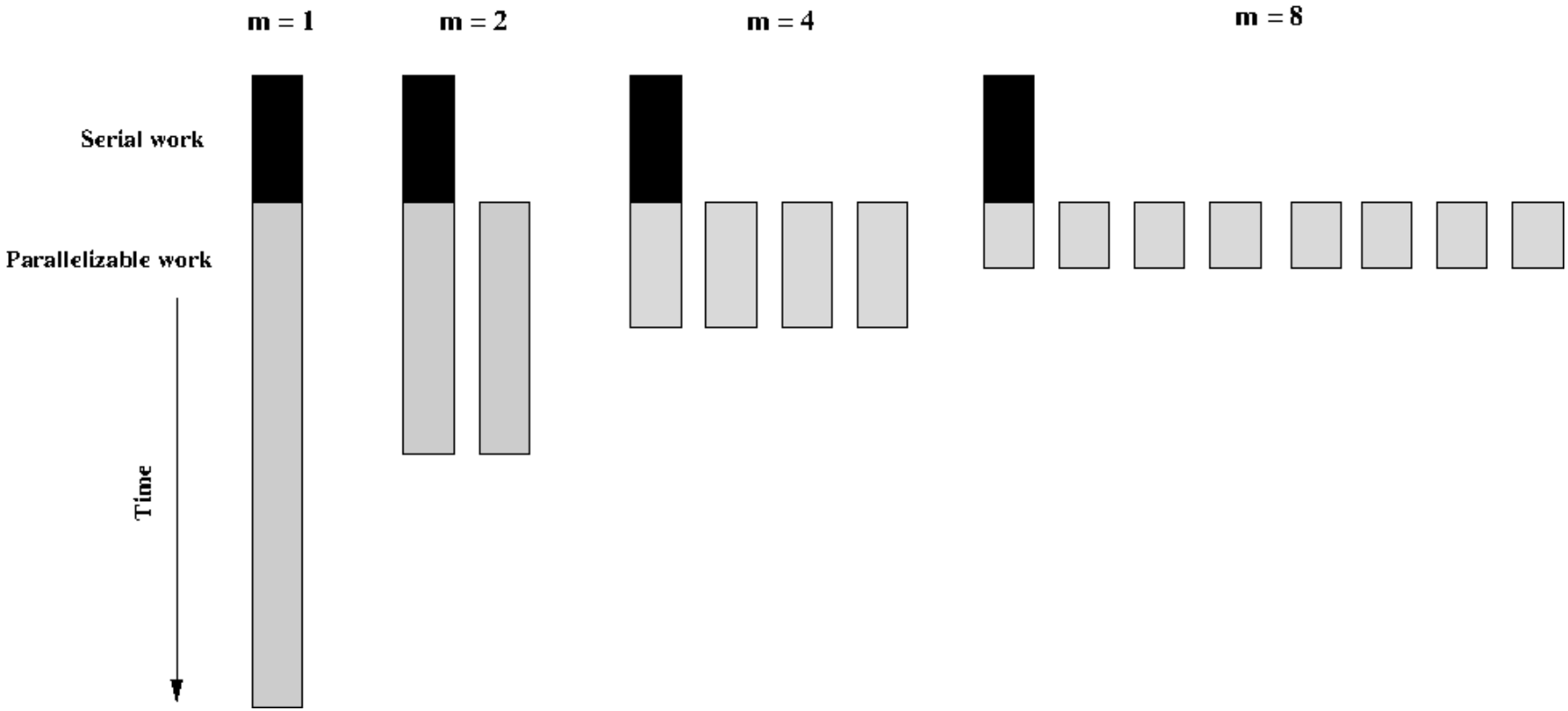
Fondamentaux du calcul parallèle - Algorithme

POINT n°1 : un algorithme parallèle *qui fait quoi quand ?*

1. identifier les tâches indivisibles
   exple : `acheter le pain` / `couper le pain` / `faire sandwichs` / `manger` / `passer le balai`
2. lister les dépendances de ces tâches, *i.e* leur ordre relatif
3. associer unités de calcul et tâches à réaliser, en respectant les dépendances
4. gérer les lectures/écritures de données communes

Fondamentaux du calcul parallèle - Algorithme

Attention, l'accélération ou *speedup* est limitée par la partie séquentielle de l'exécution (loi d'Amdahl) !



exple : `acheter le pain`

Fondamentaux du calcul parallèle - Architecture

POINT n°2 : une machine

Unité de calcul : l'objet global qui permet de réaliser l'exécution des programmes sur un ordinateur.

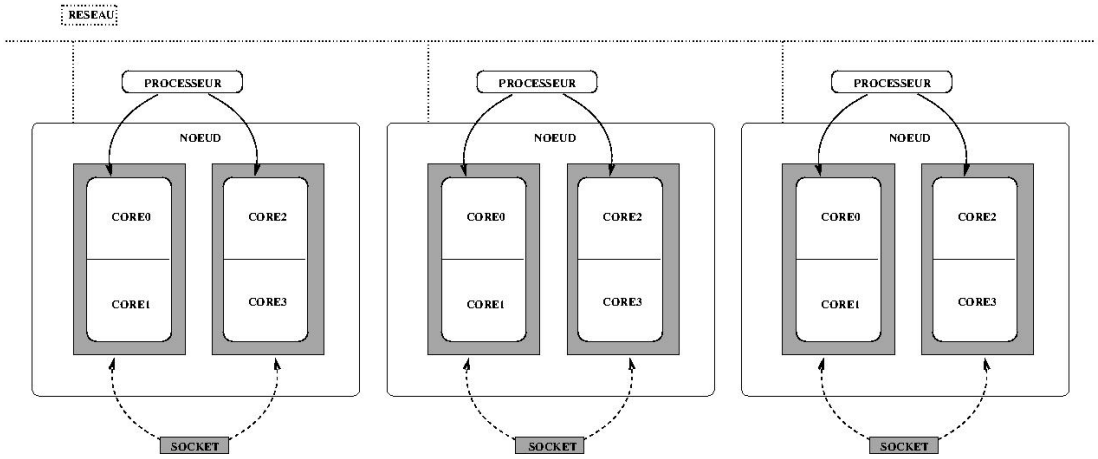
Processeur : l'ensemble des composants en charge de l'exécution des instructions machines, et donc des programmes.

Processeurs multi-cœur : il intègre plusieurs unités de calcul, les *cœurs*.

Fondamentaux du calcul parallèle - Architecture

Machine simple multi-cœurs : un ou plusieurs processeurs comprenant chacun plusieurs cœurs de calcul & mémoire ±accessible par tous les coeurs.

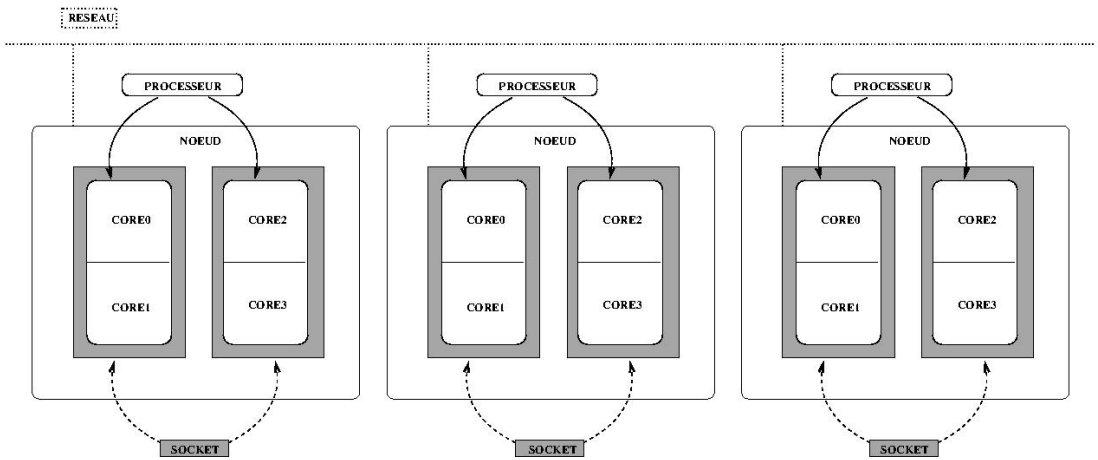
Cluster de calcul *multi-nœuds* : troupeau de plusieurs machines appelés *nœuds de calcul* & un réseau ±rapide pour communiquer



Fondamentaux du calcul parallèle - Architecture

Machine simple multi-cœurs : un ou plusieurs processeurs comprenant chacun plusieurs cœurs de calcul & mémoire ±accessible par tous les coeurs.

Cluster de calcul *multi-nœuds* : troupeau de plusieurs machines appelés *nœuds de calcul* & un réseau ±rapide pour communiquer



Kif-kif ? l'exemple du puzzle.

R parallèle sur machine multi-coeurs - Principe

Supposons que m unités de calcul sont à disposition

1. un processus R *père* est lancé et servira de chef d'orchestre ;
2. m processus R fils sont démarrés ;
3. le processus père découpe le travail et les données associées (si il y en a) et le répartit entre les processus fils ;
4. le processus père collecte les résultats du travail réalisé par les fils, puis éventuellement réalise une synthèse globale ;
5. le père arrête les processus R fils et retourne le résultat global.

R parallèle sur machine multi-coeurs - Principe

Le package `parallel` est inclus dans `R` depuis sa version `2.14.0` (31 octobre 2011). Il propose un ensemble de fonctionnalités pour introduire du parallélisme dans les codes `R` .

`parallel` est dérivé des packages CRAN `multicore` (2009-) et `snow` (2003-).

`parallel` propose des versions parallèles de `*apply` : `parApply`, `parLapply`, `parSapply`, `parVapply`.

R parallèle sur machine multi-coeurs - Principe

`parallel` propose des versions parallèles de `*apply` : `parApply`, `parLapply`, `parSapply`, `parVapply`.

NB : il y a des subtilités selon les OS

Linux&MacOSX

Windoz

R parallèle sur machine multi-coeurs - Principe

`parallel` propose des versions parallèles de `*apply` : `parApply`, `parLapply`, `parSapply`, `parVapply`.

NB : il y a des subtilités selon les OS

Linux&MacOSX

Windoz

NB : attention au générateur de nombres pseudo-aléatoires !

R parallèle sur machine multi-coeurs - Equilibrage de charge

La clé de la performance d'un code parallèle R est la répartition équilibrée des tâches à réaliser entre les processus fils, de façon à minimiser les *temps d'attente*

→ on parle alors d'*ordonnancement* de tâches
(imaginons que l'on se met à plusieurs pour repeindre une pièce biscornue...)

R parallèle sur machine multi-coeurs - Equilibrage de charge

La clé de la performance d'un code parallèle R est la répartition équilibrée des tâches à réaliser entre les processus fils, de façon à minimiser les *temps d'attente*

→ on parle alors d'*ordonnancement* de tâches
(imaginons que l'on se met à plusieurs pour repeindre une pièce biscornue...)

→ ordonnancement statique* vs dynamique, que choisir ?

(*) `mclapply` : méthode *round-robin* 

C++ parallèle dans R sur machine multi-coeurs - Principe

Le parallélisme peut se trouver au niveau du code R et au niveau des parties implémentées en langage compilé (C++ ici)

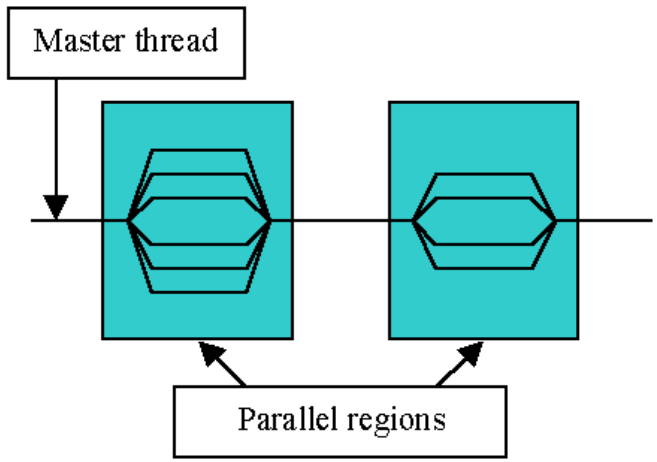
```
# code R
for (q in 1:Q){
    fonction_Rcpp(q, params)
}

// code C++
fonction_Rcpp(params){
    for (int i=0; i<N; i++){
        ... // plein de calcul
    }
}
```

C++ parallèle dans R sur machine multi-coeurs - Principe

On introduit du parallélisme en *mémoire partagée* au niveau du C++ appelé par Rcpp

→ lancer des sous-processus appelées *processus légers* ou *threads* qui exécuteront en parallèle un ensemble d'instructions indépendantes.



NB : le processus R père (pas de fils) mettra en place les différents threads (unique processus R qui tourne, mais jusqu'à $m \times 100\%$)

C++ parallèle dans R sur machine multi-coeurs - openMP

OpenMP incontournable et supporté par la grande majorité des compilateurs

Principe simple : saupoudrer le code C++ de directives (*i.e.* des mots clés) pour aider le compilateur à paralléliser tout seul

C++ parallèle dans R sur machine multi-coeurs - openMP

La directive la plus simple :

```
#pragma omp parallel
{
#pragma omp for
for (int i=0; i<10; i++)
  A[i] = B[i]*A[i];
}
```

C++ parallèle dans R sur machine multi-coeurs - openMP

La directive la plus simple :

```
#pragma omp parallel
{
#pragma omp for
for (int i=0; i<10; i++)
A[i] = B[i]*A[i];
}
```

```
#pragma omp parallel
{
#pragma omp for
for (int i=0; i<10; i++)
A[i] = B[i]*A[i-1];
// CATASTROPHE
}
```

R parallèle sur un cluster - Architecture

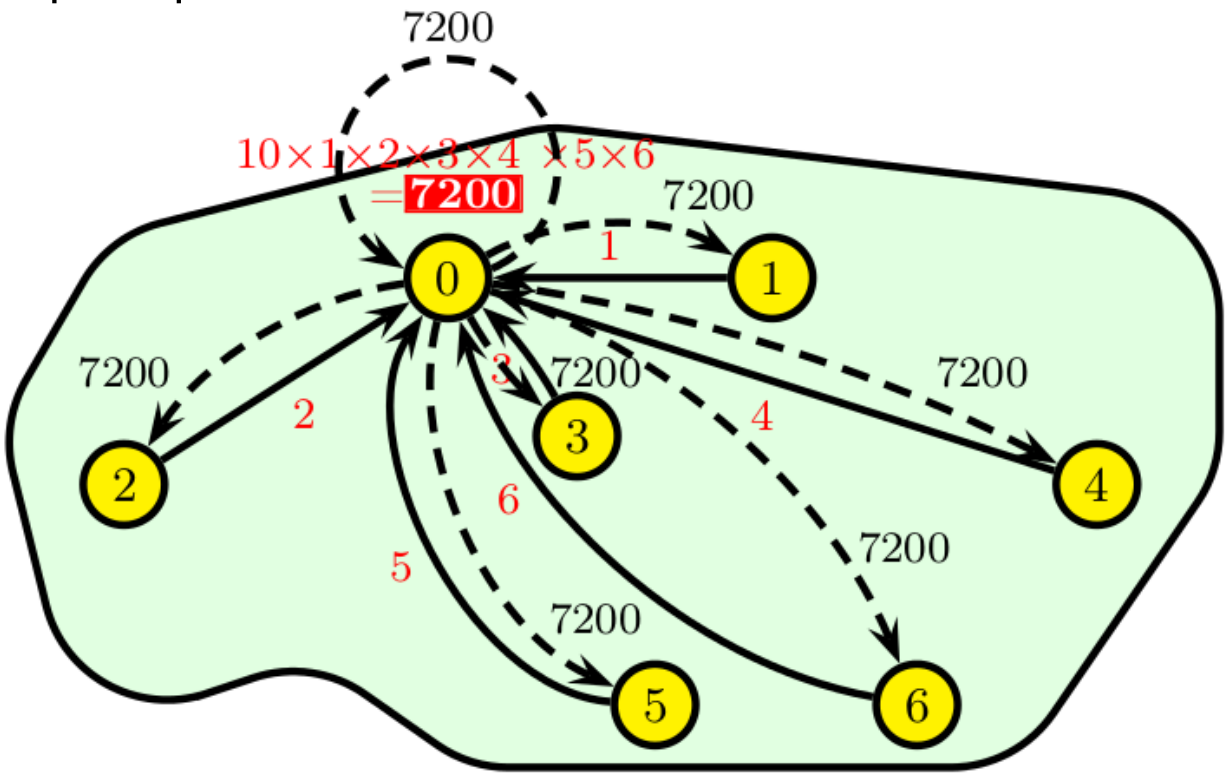
Comment peut-on tirer profit d'un cluster (sans parler du calcul réparti) ?

- les clusters dédiés au *calcul intensif* et au *HPC* (*High Performance Computing*) ;
- les clusters dédiés au *calcul sur données distribuées* et au *big data* (données trop volumineuses pour un traitement classique).

R parallèle sur un cluster - MPI

Avec `Rmpi` et `pbdMPI`.

Le principe fondamental de `MPI` est le suivant :



R parallèle sur un cluster - MPI

```
## récupération centralisée des résultats
if(.comm.rank == 0){
    for (rank in 1:(.comm.size-1)){
        ## réception du res de chaque processus de rang >0
        res <- recv(res, rank.source=rank)
    }
    print(total.res)
} else{
    ## envoi de res vers le processus 0
    send(res, rank.dest=0, tag=.comm.rank)
}
```

R parallèle sur un cluster - Hadoop/Spark

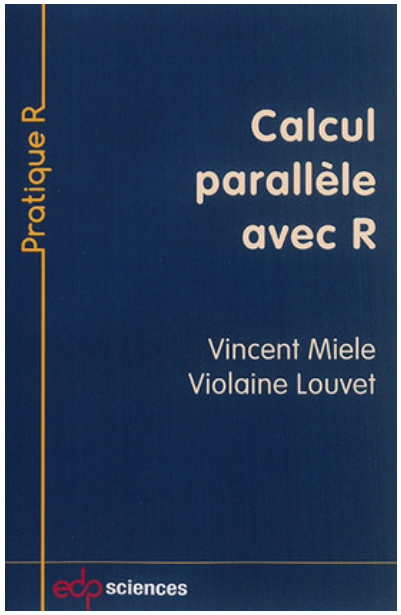
Hadoop : les données sont directement stockées de manière distribuée sous forme de blocs répartis sur les différents noeuds `RHadoop`

MapReduce : un algorithme de décomposition itérative de décomposition/fusion

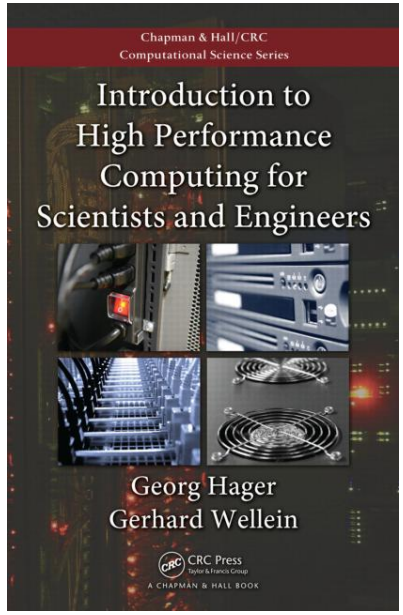
`Spark` : une alternative « in-memory » (c'est à dire qui exploite la mémoire plutôt que le disque) `SparkR`

Bilan

Le calcul parallèle pour non-spécialistes, c'est maintenant ?



O



U

I

vincent.miele@univ-lyon.fr
http://calcul.math.cnrs.fr