Calcul parallèle avec R
ANF R

Vincent Miele
CNRS

07/10/2015

Pour chaque exercice, il est nécessaire d'ouvrir une fenètre de visualisation des processes (`Terminal + top` sous Linux et Mac OS X, `Gestionnaire des tâches` sous Windows) et d'observer le comportement des processes `R`.
Par ailleurs, si votre machine a plus que 4 GB de RAM, remplacer dans ce qui suit `m=3000` par `m=5000`.
Enfin, avant chaque exercice, il est préférable de relancer `R` (sans sauvegarde de l'environnement).

Exercice 1 (Parallel apply)

1. tester la version séquentielle

```
n=1000
m=30000
M = matrix(rnorm(n*m), ncol=m)
V = matrix(rnorm(10*m), ncol=m)

eucd <- function(u){
euc.distu = apply(V, 1, function(w) sqrt(sum((u - w)^2)))
}

gc()
ptm <- proc.time()
euc.dist1 = apply(M, 1, eucd)
proc.time() - ptm
```

2. tester la version *snow-like*

```
library(parallel)

gc()
cl <- makeCluster(2, type="PSOCK")
clusterExport(cl, list("V"))
ptm <- proc.time()
euc.dist2 = parApply(cl, M, 1, eucd)
proc.time() - ptm
stopCluster(cl)
```

3. tester la version *multicore-like*

```
gc()
```

```
cl <- makeCluster(2, type="FORK")
ptm <- proc.time()
euc.dist3 = parApply(cl, M, 1, eucd)
proc.time() - ptm
stopCluster(cl)
```

Exercice 2 (Parallel lapply)

1. tester la version séquentielle

   ```
   n=1000
   m=30000
   L = lapply(1:n, rnorm, n = m )
   V = matrix(rnorm(10*m), ncol=m)


   eucd <- function(u){
   euc.distu = apply(V, 1, function(w) sqrt(sum((u - w)^2)))
   }

   gc()
   ptm <- proc.time()
   euc.dist1 = lapply(L, eucd)
   proc.time() - ptm
   ```

2. tester tester la version *snow-like*

   ```
   library(parallel)
   gc()
   cl <- makeCluster(2, type="PSOCK")
   clusterExport(cl, list("V"))
   ptm <- proc.time()
   euc.dist2 = parLapply(cl, L, eucd)
   proc.time() - ptm
   stopCluster(cl)
   ```

3. tester tester les deux versions *multicore-like*

   ```
   gc()
   ptm <- proc.time()
   euc.dist1 = mclapply(L, eucd, mc.cores=2)
   proc.time() - ptm

   gc()
   cl <- makeCluster(2, type="FORK")
   ptm <- proc.time()
   euc.dist3 = parLapply(cl, L, eucd)
   proc.time() - ptm
   stopCluster(cl)
   ```

Exercice 3 (mcparallel et mccollect)
Tester le lancement de tâches (Linux et Mac OS X seulement)

1.
   ```
   library(parallel)
   library(igraph)

   mis <- function(g){
   ptm <- proc.time()
   lis = largest.independent.vertex.sets(g)[1]
   t = proc.time() - ptm
   cat("Graph with ",vcount(g)," vertices : DONE in ",as.numeric(t[3]),"\n")
   lis
   }

   g1 <- erdos.renyi.game(70, 1/25)
   mis1 = mcparallel(mis(g1))
   mccollect(mis1, wait=FALSE)

   g2 <- erdos.renyi.game(60, 1/25)
   mis2 = mcparallel(mis(g2))
   mccollect(list(mis1, mis2), wait=TRUE)
   ```

Exercice 4 (Parallel foreach)

1. tester la version séquentielle

   ```
   library(foreach)
   n=1000
   m=30000
   M = matrix(rnorm(n*m), ncol=m)
   V = matrix(rnorm(10*m), ncol=m)

   ptm <- proc.time()
   euc.dist4 = foreach(i=1:n, .combine=c) %do% eucd(M[i,])
   proc.time() - ptm
   ```

2. tester tester la version *snow-like*

   ```
   library(foreach)
   library(parallel)
   library(doParallel)
   cl <- makeCluster(2)
   registerDoParallel(cl)
   ptm <- proc.time()
   euc.dist6 = foreach(i=1:n, .combine=c) %dopar% eucd(M[i,])
   proc.time() - ptm
   stopCluster(cl)
   ```

3. tester tester la version *multicore-like*

   ```
   registerDoParallel(cores=2)
   ptm <- proc.time()
   euc.dist5 = foreach(i=1:n, .combine=c) %dopar% eucd(M[i,])
   proc.time() - ptm
   ```

Exercice 5 (Scheduling)

1. un peu d'initialisation…

   ```
   library(parallel)

   lazy <- function(i){
   Sys.sleep(i)
   cat(Sys.getpid()," waited ",i," seconds\n")
   Sys.getpid()
   }

   L = list(5,30,5,10,5,10)
   ```

2. comparer les 2 versions *multicore-like* (Linux et Mac OS X)

   ```
   ptm <- proc.time()
   res = mclapply(L, lazy, mc.cores=2) # preschedule 1 by 1
   proc.time() - ptm
   res

   ptm <- proc.time()
   res = mclapply(L, lazy, mc.cores=2, mc.preschedule = FALSE)
   proc.time() - ptm
   res
   ```

3. comparer les 2 versions *snow-like*

   ```
   cl <- makeCluster(2)
   clusterSplit(cl, L)
   ptm <- proc.time()
   res = parLapply(cl, L, lazy) # preschedule by chunks
   proc.time() - ptm
   stopCluster(cl)
   res

   cl <- makeCluster(2)
   ptm <- proc.time()
   res = parLapplyLB(cl, L, lazy) # fake, watch the source
   proc.time() - ptm
   stopCluster(cl)
   res
   ```

4. reprendre avec une "vrai" function `parLapplyLB` définie sur `http://examples.oreilly.com/0636920021421/snow/loadbalancing.R`

   ```
   parLapplyLB <- function(cl, x, fun, ...) {
   clusterCall(cl, LB.init, fun, ...)
   r <- clusterApplyLB(cl, x, LB.worker)
   clusterEvalQ(cl, rm('.LB.fun', '.LB.args', pos=globalenv()))
   r
   }
   LB.init <- function(fun, ...) {
   assign('.LB.fun', fun, pos=globalenv())
   assign('.LB.args', list(...), pos=globalenv())
   ```

```
NULL
}
LB.worker <- function(x) {
do.call('.LB.fun', c(list(x), .LB.args))
}
```

Exercice 6 (Scheduling for real)

1. un peu d'initialisation... (notamment tirage aléatoire de graphes)

   ```
   library(parallel)
   library(igraph)

   L = lapply(list(50,10,50,25,50,25), function(i){g <- erdos.renyi.game(70, 1/i)})

   mis <- function(g){
   ptm <- proc.time()
   lis = largest.independent.vertex.sets(g)[1]
   t = proc.time() - ptm
   cat("Graph with ",vcount(g)," vertices : DONE in ",as.numeric(t[3]),"\n")
   lis
   }
   ```

2. comparer les 2 versions *multicore-like* (Linux et Mac OS X)

   ```
   ptm <- proc.time()
   res = mclapply(L, mis, mc.cores=2)
   proc.time() - ptm

   ptm <- proc.time()
   res = mclapply(L, mis, mc.cores=2, mc.preschedule = FALSE)
   proc.time() - ptm
   ```

3. comparer les 2 versions *snow-like*

   ```
   cl <- makeCluster(2)
   ptm <- proc.time()
   clusterEvalQ(cl, {library(igraph); NULL})
   res = parLapply(cl, L, mis) # preschedule by chunks
   proc.time() - ptm
   stopCluster(cl)

   cl <- makeCluster(2)
   ptm <- proc.time()
   clusterEvalQ(cl, {library(igraph); NULL})
   res = parLapplyLB(cl, L, mis) # fake, watch the source
   proc.time() - ptm
   stopCluster(cl)
   ```

4. éventuellement reprendre depuis le début de l'exercice pour tirer de nouveaux graphes

Exercice 7 (clusterExport)
Tester la présence et l'absence du `clusterExport`

```
library(parallel)

n=1000
m=30000
M <- matrix(rnorm(n*m), ncol=m) # M is in GlobalEnv
cl <- makeCluster(2, type="PSOCK")
doSleep1 <- function(i) {Sys.sleep(i); M[1,1]=0;}
# clusterExport(cl, list("M"))
res = parLapply(cl, list(20,20), doSleep1)
stopCluster(cl)
```

Exercice 8 (Serialization et mal de tête)

1. après avoir relancé `R`, observer la consommation mémoire des processes au fur et à mesure de l'execution de la fonction `parSleep`. Comprendre le contenu des fichiers de sorties `output_xxx.txt`.

```
library(parallel)

parSleep <- function(){
  n=1000
  m=30000
  M <- matrix(rnorm(n*m), ncol=m) # M is not in GlobalEnv
  # then clusterExport is impossible
  M2 <- matrix(rnorm(n*m), ncol=m)
  gc()

  doSleep1 <- function(i) {
    library(pryr)
    fileConn<-file(paste("output_",Sys.getpid(),".txt",sep=""))
    writeLines(paste("Memory in doSleep1 :",mem_used()), fileConn)
    close(fileConn)
    Sys.sleep(i);
  } # without, M and M2 will be serialized

  doSleep2 <- function(i) {
    library(pryr)
    fileConn<-file(paste("output_",Sys.getpid(),".txt",sep=""))
    writeLines(paste("Memory in doSleep1 :",mem_used()), fileConn)
    close(fileConn)
    Sys.sleep(i);
  }
  environment(doSleep2) <- .GlobalEnv # without, M and M2 would be serialized

  doSleep3 <- function(i, M) { # M in parameter
    library(pryr)
    fileConn<-file(paste("output_",Sys.getpid(),".txt",sep=""))
    writeLines(paste("Memory in doSleep1 :",mem_used()), fileConn)
    close(fileConn)
    Sys.sleep(i);
```

```
      M[1,1]=0;
    }
    environment(doSleep3) <- .GlobalEnv # without, M is serialized but M2 too

    cl <- makeCluster(2, type="PSOCK")
    cat("doSleep1\n")
    res = parLapply(cl, list(20,20), doSleep1)
    stopCluster(cl)

    cl <- makeCluster(2, type="PSOCK")
    cat("doSleep2\n")
    res = parLapply(cl, list(20,20), doSleep2)
    stopCluster(cl)

    cl <- makeCluster(2, type="PSOCK")
    cat("doSleep3\n")
    res = parLapply(cl, list(30,30), doSleep3, M)
    stopCluster(cl)
    NULL
  }

  parSleep()
```

2. reprendre la question précédente (dont la relance de `R`) en enlevant la/les ligne(s) `environment(doSleep...` et comparer