Sweave en parallèle

Ou "*Comment j'ai réussi à produire 14 rapports différents en faisant fonctionner les cœurs de mon ordinateur plutôt que ma pauvre tête...*"

Christophe Bontemps[1]

Résumé. Je rapporte ici une expérience de création automatique de 14 rapports issus d'une même structure, comprenant les mêmes objets de recherche mais portant sur des fichiers de données différents. Je montre qu'il est très facile de mettre en œuvre des outils permettant de répéter des travaux de recherche empiriques et qu'il est réellement simple d'exécuter ces tâches en parallèle sous R. Les gains en performance sont très appréciables, y compris sur une machine de bureau, tandis que les coûts de mise en œuvre et d'apprentissage sont très faibles, comme l'illustre cette expérience.

Mots clés : R, *Sweave* , parallélisme, génération automatique de documents, recherche reproductible

Introduction : une histoire de fainéant

L'histoire commence par un travail de recherche mené dans le cadre d'un projet financé par l'Agence Nationale de la Recherche (ANR) sur l'efficacité et la productivité des industries agro-alimentaires. Nous disposons pour ce projet de données sur un grand nombre de secteurs agroindustriels (viandes, lait, etc.) identifiés par un code représentant le secteur de leur activité principale. Après sélection, 14 secteurs - soit 14 fichiers - sont retenus sur lesquels nous allons conduire une analyse complète. Nous décidons de mettre au point notre analyse économétrique et nos programmes sur un secteur et de répliquer cette analyse sur les autres. Le point de départ est donc clairement de créer une procédure générique permettant d'examiner les résultats de nos estimations sur 14 secteurs, soit 14 fichiers de données, afin de créer 14 rapports. Sachant que le programme initial sera amené à évoluer en cours de route, que certaines estimations sont assez longues (on effectuera par la suite du bootstrap, nécessitant de nombreuses réplications) et que l'on cherche la plus grande souplesse dans l'utilisation et la modification des outils économétriques, le choix de l'environnement de travail s'est porté assez naturellement sur le tandem *R* et *Sweave* (Leisch, 2002).

Le choix de Sweave… et de LaTeX

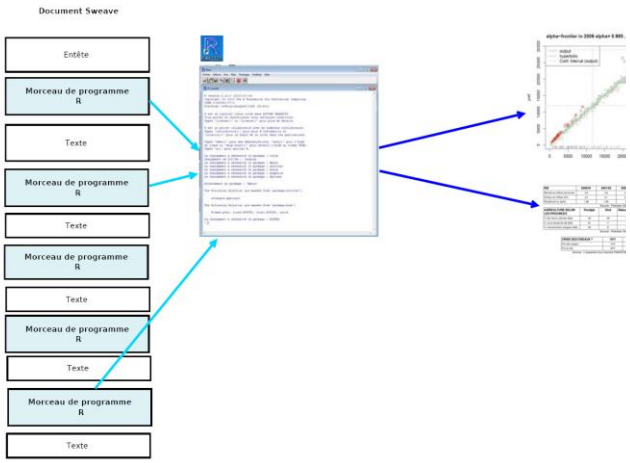
LaTeX (Lamport, 1995) est un langage de mise en forme de documents permettant l'écriture en mode texte de commandes permettant la composition d'un document structuré (sections, équations, tables, etc.). À la manière d'Html, le code de mise en forme entoure littéralement le texte lui-même. Depuis de nombreuses années LaTeX est extrêmement populaire pour l'écriture d'articles scientifiques. *Sweave* est un package de R qui permet à la fois de structurer un document et d'exécuter des morceaux ("*chunks*") de code R. Un programme Sweave est

[1]Toulouse School of Economics, UMR 1291, Gremaq-INRA, 21 allée de Brienne, F-31000 Toulouse, France. Christophe.Bontemps@toulouse.inra.fr

donc un document LaTeX[2] un peu particulier (extension *.Rnw*). Lors de l'appel de Sweave sur un fichier les différentes sections contenant du code R sont exécutées et, suivant les options spécifiées pour chaque morceau, les résultats sont incorporés (ou pas) dans le document LaTeX final. On peut ainsi intégrer dans un document tables, graphiques et sorties de n'importe quelle commande R (Meredith et Racine, 2009). L'utilisation de *Sweave* demande donc une compétence initiale en R et en LaTeX, mais permet de s'affranchir de tout copier/coller et est donc parfaitement adaptée à une démarche de recherche reproductible (Koenker et Zeileis 2009). En outre, le document produit est hautement interactif puisque modifiable, soit dans le texte, soit dans les sorties des programmes en effectuant des modifications du code R incorporé, par une simple réexécution de la commande Sweave (Nature, 2013).

Le processus de création du document final est représenté dans les **Figures 1a-1d**. On représente schématiquement le document source (.Rnw) dans la **Figure 1a**. Il est composé de morceaux de code R (en bleu) insérés dans un document LaTeX. Ces morceaux sont ensuite traités par R et diverses sorties sont créés de manière temporaire ou pas (**Figure 1b**). Suivant les options choisies pour chaque morceau, ces sorties sont intégrées automatiquement dans un document LaTeX. (**Figure 1c**), qui est ensuite compilé pour donner naissance au rapport final en pdf (**Figure 1d**). Le tout sans aucune intervention humaine, ni sur les calculs, ni sur les résultats, ni sur les noms des graphiques, tables, résultats, ni sur l'intégration de ces différents éléments dans le fichier final. Cette opération permet donc d'assurer la traçabilité des traitements qui sont intégralement incorporés dans le code source du document ayant engendré le rapport final (Bontemps et Orozco, 2013).

(1a) Fichier MonAnalyse.Rnw



(1b) Exécution des blocks par R

[2]Sweave permet aussi de travailler avec des documents *OpenOffice Writer* (Leisch, 2002).

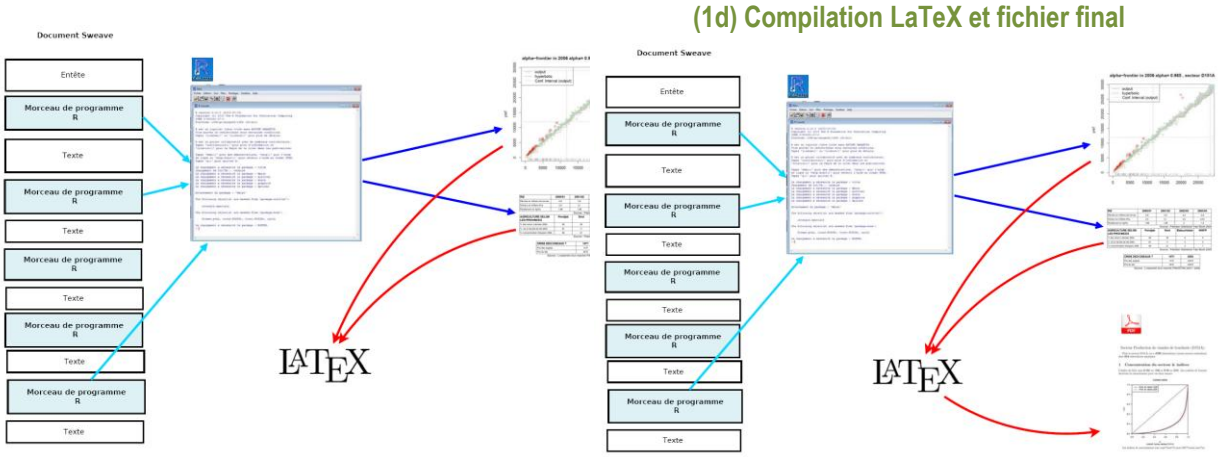


Figure 1. *Etapes du traitement d'un fichier par Sweave.*

Notre premier travail a donc été de mettre au point nos programmes et notre code R, de sélectionner les tables, graphiques et sorties adaptées à nos besoins et utiles à notre recherche afin de créer un programme Sweave, que nous nommerons *MonAnalyse.Rnw*, suffisamment générique pour qu'il s'applique à n'importe lequel des fichiers de nos secteurs agroalimentaires. Le plus dur là-dedans est évidement de construire le programme de recherche et de mettre au point les outils économétriques dans R, mais tout ceci est facilité par l'utilisation d'un environnement de travail adapté[3] et par une bonne connaissance des multiples possibilités de R.

Une boucle sur Sweave ?

Une idée vient évidemment à l'esprit de toute personne soucieuse de son temps, consciente de la lourdeur d'une répétition de ce processus 14 fois et voulant éviter toute possibilité d'erreur d'une telle entreprise : si l'on peut créer un rapport en un clic avec Sweave, est- il possible d'automatiser le traitement pour créer plusieurs documents ? En résumé : peut-on faire une boucle ?

La commande Sweave n'est pas conçue pour que ses fichiers prennent un quelconque paramètre en entrée[4]. En effet, la commande que R exécute est `Sweave("MonAnalyse.Rnw")` et ne permet pas cela directement. En outre, comme nous l'avons vu, le fichier Sweave est un document texte formaté et n'est pas un programme R, Il faut donc réfléchir un peu.

Une idée …

S'il n'est pas possible de créer directement une boucle qui exécuterait séquentiellement Sweave sur différents fichiers, il est cependant facile de contourner l'obstacle. Il suffit en effet d'une petite modification toute bête pour intégrer un paramètre directement dans notre fichier Sweave. Ainsi, dans le programme *MonAnalyse.Rnw*, au lieu de préciser directement "en dur" le code/nom de référence du fichier de données sur lequel s'appliquera

[3]Un environnement de travail particulièrement adapté au travail avec *Sweave* est *RStudio* où l'appel de la commande *Sweave* est simplifié et intégré (Racine, 2012).

[4] Il existe un autre package R, nommé *Brew*, qui permet d'exécuter différentes commandes et de les répéter sur différents fichiers. Cependant, ce package ne permet pas (ou pas facilement) de répondre à la problématique proposée ici et n'est pas utilisable dans un cadre de parallélisme.

notre programme, nous allons transformer ce code/nom en paramètre et importer ce code/nom depuis un fichier texte annexe nommé *Fichiers.dat*. Par cette externalisation du code/nom de fichier, le programme Sweave générique *MonAnalyse.Rnw* dépend maintenant d'un paramètre extérieur qu'il devra donc lire dans un fichier annexe. Si l'on souhaite exécuter ce programme Sweave sur différents fichiers, nous pouvons modifier *Fichiers.dat* et y écrire (ou faire écrire par un programme extérieur), notre paramètre (le code/nom du fichier de données, donc) sans modifier en rien le fichier *MonAnalyse.Rnw.*

On pourra ainsi modifier le code du premier chunk R du fichier *MonAnalyse.Rnw*, en supprimant l'appel explicite à un nom de fichier de données et en le remplaçant par la lecture des éléments du fichier *Fichiers.dat*. On utilise pour cela la fonction "scan()" et on s'assurera que le nom du fichier de données est bien du format chaîne de caractères ("as.character()"). Le code du premier *chunk* change[5] donc de :

```
<<echo=FALSE, results=HIDE>>=
# RECUPERATION du Code/Nom du fichier de données
Nom <- "D151C"
@
```

à :

```
<<echo=FALSE, results=HIDE>>=
 # RECUPERATION du Code/Nom du fichier de données
foo <-scan("Fichiers.dat", what = list(""))
Nom <- as.character(foo[1])
@
```

Le reste du document *MonAnalyse.Rnw* est inchangé. Cette petite manipulation offre bien des avantages, comme nous le verrons par la suite puisqu'elle permet, sans altération du fichier Sweave, de résoudre notre problème.

Sweave séquentiel

Dès lors *MonAnalyse.Rnw* récupère le nom du fichier de données à utiliser en lisant directement le contenu de *Fichiers.dat*. On pourra alors modifier indépendamment le contenu unique de ce fichier (le nom du fichier de données) pour changer et donc paramétrer le fichier de données utilisé pour l'analyse. Il devient alors facile de créer un programme d'exécution séquentiel (nommé *SequentielSweave.R*) sur l'ensemble des fichiers de données. Ce programme maître pourra modifier séquentiellement le fichier externe[6] *Fichiers.dat* et ainsi permettre une exécution du programme Sweave sur *MonAnalyse.Rnw* pour différents fichiers de données, sans avoir à modifier le contenu de ce programme générique. On s'assurera néanmoins de concevoir le programme d'exécution séquentielle de manière à ne pas écraser un rapport créé pour un fichier de données par son successeur dans la liste, c'est à dire en gérant convenablement l'arborescence et le nom des fichiers en sortie par exemple. L'arborescence pourra ressembler à celle présentée **Figure 2**, où chaque rapport propre à chaque fichier de données est traité et compilé dans un répertoire spécifique. Ici les noms des fichiers commencent par la lettre D suivie de 3 chiffres et d'une lettre (ex : D151C, D155A, etc.).

---

[5]Dans la syntaxe de *Sweave*, les chunks sont délimités par les balises <<>> (début) et @ (fin), les options apparaissant également entre les << et >>. Le chunk ci-dessous n'affiche donc rien (echo=FALSE) et ne reporte aucun affichage de résultat (results=HIDE). A noter que les lignes précédées d'un # sont des commentaires en R.

[6]On peut exécuter des programmes externes depuis R, par exemple le shell (Dos ou Linux).

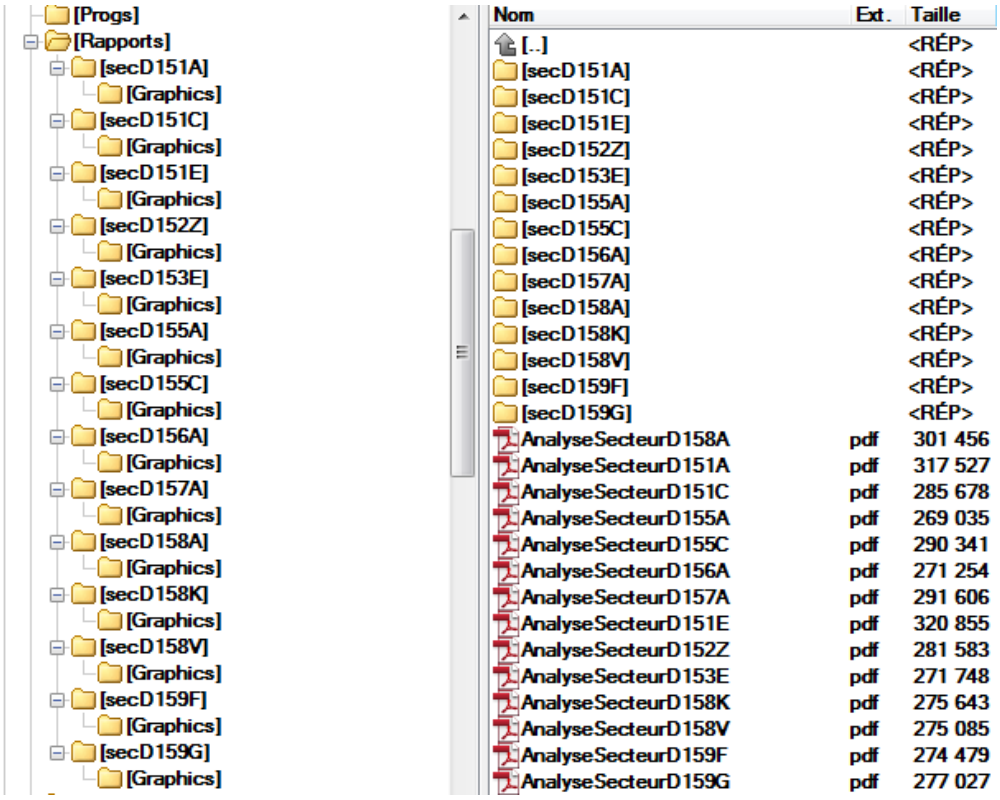


Figure 2. *Arborescence finale*

Le pseudo-code ou algorithme d'un programme réalisant l'ensemble de ces opérations séquentiellement pour chaque secteur est le suivant.

Pour chaque élément de la liste des fichiers de données, FAIRE :

- MODIFIER le fichier externe *Fichiers.dat* avec le nom du fichier de données courant,
- CRÉER une arborescence propre au secteur,
- EXÉCUTER Sweave sur le fichier de données récupéré du fichier externe, dans le répertoire spécifique à ce fichier de données (création des graphiques et tables)
- EXÉCUTER le programme LaTeX, (création du rapport pdf spécifique).

Passer au fichier de données suivant

La séquence séquentielle décrite dans ce pseudo-code peut être représentée schématiquement par l'enchaînement décrit dans les **Figures 3(a)-3(d)**.

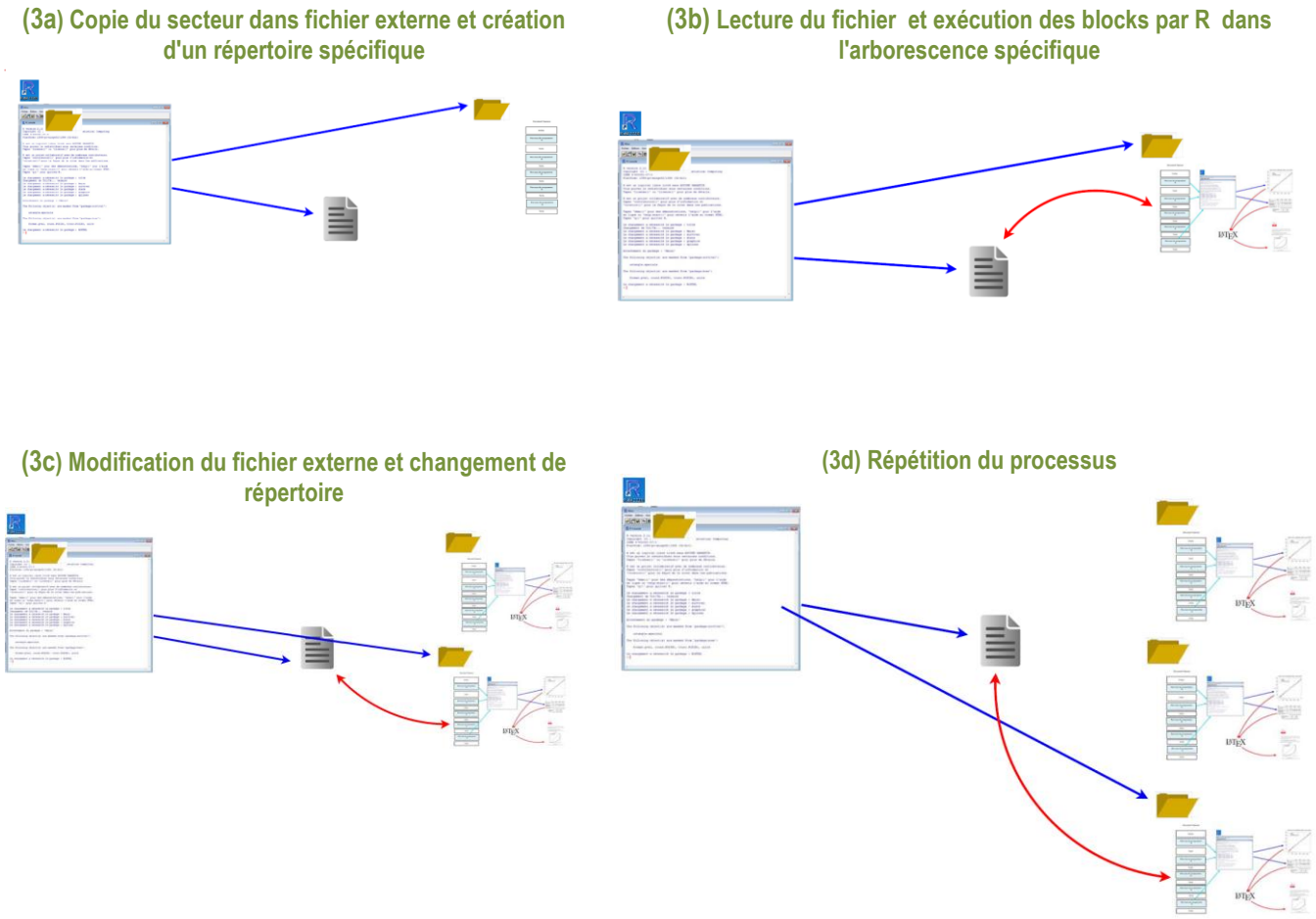


Figure 3. *Exécution séquentielle du fichier Sweave sur différents fichiers de données (le fichier maître est représenté en haut à gauche et ordonne les opérations).*

Code du programme maître séquentiel en R

Le programme suivant, écrit en R[7] , qui sera le programme "maître" de l'exécution utilise 3 fonctions de R pour : **écrire** dans un fichier externe ("write()"), **exécuter** Sweave ("utils::Sweave()") et **compiler** le document LaTeX créé ("tools::texi2dvi()") ainsi que des fonctions inspirées du shell (MS-Dos) pour créer des répertoires et copier des fichiers ("dir.create()" et "file.copy()"). On utilise le nom du s^{ième} fichier de données (en le récupérant dans la liste FichList[s]) pour créer des répertoires spécifiques à ce fichier de données pour nos fichiers intermédiaires (et comme suffixes de nos rapports finaux). Le code de ce programme est ci-dessous :

[7]Il est également possible de trouver d'autres solutions, comme celle d'écrire directement dans le fichier *MonAnalyse.Rnw*, à la bonne ligne, au bon endroit à l'aide d'un programme externe comme Python par exemple. On peut également envisager l'utilisation d'un *makeFile* (Laurent, 2013).

```
#Ensemble des noms des fichiers de données
 FichList <-c("D151E", "D152Z", "D153E", "D158K", "D158V",
                "D159F", "D159G", "D151A", "D151C", "D155A",
                "D155C", "D156A", "D157A", "D158A")

for (s in 1:length(FichList)) {

        #On nourrit le fichier servant pour le nom du fichier de données
        write(FichList [s], file="Fichiers.dat", append=FALSE)

        #On execute Sweave + texi2pdf

        utils::Sweave("MonAnalyse.Rnw")
        tools::texi2dvi("MonAnalyse.tex", clean = TRUE, pdf = TRUE)

        #On créée et on remplit les répertoires des fichiers sources .tex, etc...

        dir.create(paste("sec", FichList [s], sep =""), showWarnings = FALSE)
        file.copy("Graphics", paste("sec", FichList [s], sep =""), recursive=TRUE)

        #On duplique les rapports avec en suffixe le nom du secteur (optionnel)

        file.copy("MonAnalyse.pdf", paste("sec",FichList[s],"/MonAnalyseNew",
                                          FichList[s],".pdf", sep =""), overwrite = TRUE)
} # <– fin de boucle
```

De ce programme naît donc l'arborescence présentée dans la **Figure 2** au sein de laquelle nous avons répliqué tour à tour et dans des répertoires différents, le même processus basé sur l'exécution du même programme Sweave *MonAnalyse.Rnw*. Nous verrons dans la section 4 que ce processus bien que complètement automatique, peut s'avérer très long.

Des mondes parallèles

Une fois cette étape franchie avec succès, l'idée de transformer cette exécution séquentielle des 14 rapports en une exécution parallèle s'impose à tout esprit curieux de repousser les limites de l'automatisation.[8] Il faut préciser ici que l'idée de travailler en parallèle est particulièrement adaptée puisqu'il n'y a absolument aucune interaction entre les tâches. L'exécution d'une instance de R/Sweave sur un fichier de données se passe en effet en toute indépendance de l'exécution d'une autre instance de R/Sweave sur un autre fichier de données. C'est le cas idéal pour utiliser le parallélisme et les fonctions du package snowfall de R.

Un package bien sympa

Le package snowfall de R permet d'utiliser facilement la puissance du calcul parallèle sur des fonctions R (Porzelius et al., 2009). L'un des avantages de l'approche que je présente ici réside dans sa simplicité. Je n'utiliserai en effet ici qu'une seule fonction du package snowfall de R (la fonction sfClusterApplyLB()), qui prend comme paramètres la fonction à exécuter en parallèle et le nombre de cœurs sur lequel travailler[9].

[8] Rappelons que l'exécution sur un seul fichier de données peut prendre plus d'une heure.

[9] Un petit détail technique en passant : la fonction sfClusterApplyLB()) applique un principe d'équilibrage des charges entre les cœurs (*Load Balanced*), de sorte que lorsqu'une tâche est terminée sur un cœur, une nouvelle tâche lui est affectée sans attendre.

Puisque snowfall s'applique à des fonctions de R, il me faut donc transformer le code appelé dans le corps de ma procédure séquentielle en une fonction R autonome.

Une fonction pour Sweave en parallèle La première étape consiste donc à créer une fonction qui aura pour objet d'exécuter Sweave sur le fichier *MonAnalyse.Rnw* dans une arborescence propre à un fichier de données et qui prendra donc le nom de ce fichier comme paramètre. Cette fonction s'inspire des opérations utilisées dans la boucle exécutée séquentiellement présentée plus haut. Dans une seconde étape, tout comme pour la procédure séquentielle, je créée un fichier maître R dans lequel j'initialise la parallélisation, et exécute la commande "sfClusterApplyLB()" appliquée à ma fonction sur les différents cœurs de mon ordinateur[10].

Ma fonction sweave.paral, qui reprend les principes et les outils de la procédure séquentielle est présentée ci-dessous. Cette fonction prend comme unique paramètre le nom du fichier de données utilisé. Outre l'écriture sous forme de fonction, on remarquera que le fichier annexe *Fichiers.dat* doit maintenant être recopié dans des répertoires différents, tout comme mon fichier *MonAnalyse.Rnw,* afin de pouvoir exécuter Sweave sur des fichiers de données différents, de manière indépendante. Le principe est illustré dans les **Figures 4a et 4b.**

```
# Définition de la fonction Sweave.paral
Sweave.paral = function (NomFich){

 setwd("D:/ANR-EPI/")

 #On créée et on remplit les répertoires des données et du fichier Sweave.

 dir.create(paste("Fich",NomFich, sep =""), showWarnings = FALSE)
 dir.create(paste("Fich",NomFich,"/Data", sep =""), showWarnings = FALSE)
 dir.create(paste("Fich",NomFich,"/Graphics", sep =""), showWarnings = FALSE)

 #Copie du fichier Sweave et des données (et des fonctions annexes si nécessaire)
# dans le répertoire spécifique au fichier de données

 file.copy(from="AnalyseSecteur.Rnw",
           to=paste("Fich",NomFich,"/",
                    paste("Analyse",NomFich,".rnw",sep=""),sep=""),
           overwrite=TRUE)
 file.copy(from=paste("Data","/",NomFich,"Stat.dta",sep=""),
           to=paste("Fich",NomFich,"/Data/", sep =""),
           overwrite=TRUE)

# On se place dans ce répertoire spécifique au fichier de données

setwd(paste("Rapports/Fich",NomFich, sep =""))
write(NomFich, file="Fichiers.dat", append=FALSE)


# On exécute Sweave + texi2pdf
 utils::Sweave(paste("Analyse",NomFich,".rnw",sep=""))
 tools::texi2pdf(paste("Analyse",NomFich,".tex",sep=""),  clean = TRUE)
```

[10]Dans la section 4, je propose quelques tests de performance en faisant varier le nombre de cœurs de 1 (pas de parallélisation) à 14 (parallélisation *complète*).

```
#On duplique les rapports avec en suffixe le nom du secteur (optionnel)

  file.rename(from=paste("Analyse",NomFich,".pdf",sep="") ,
              to=paste("Analyse",NomFich,".pdf",sep=""))

} # <-- fin de fonction
```

(4a) Exécution de la fonction et du fichier maître

(4b) Exécution simultanée sur différents cœurs dans différents répertoires

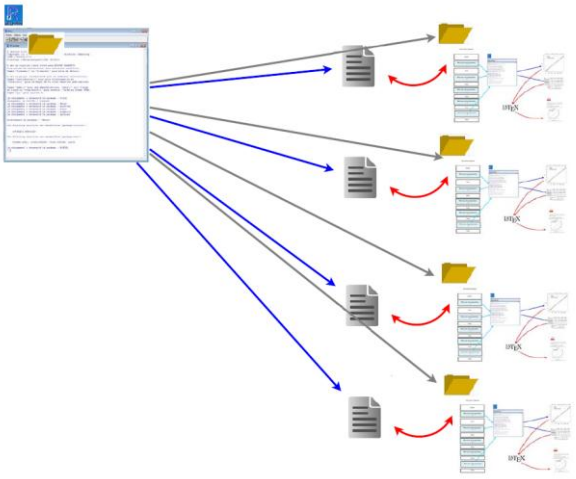


Figure 4. *Traitement en parallèle d'un fichier Sweave.*

Code du programme maître parallèle en R

Le code de mon programme "maître" permettant la parallélisation de l'exécution de mon programme Sweave *MonAnalyse.Rnw* sur 14 fichiers de données, est également très simple. Il est constitué, d'une part, de la fonction *Sweave.paral* décrite ci-dessus, suivi d'autre part, de son exécution en parallèle en utilisant les fonctions du package snowfall. Une fois cette fonction connue de R, le programme maître pourra indiquer à R de charger le package snowfall téléchargé au préalable. On précisera ensuite le nombre de cœurs utilisés. L'utilisation de la fonction "sfClusterApplyLB()" est ultra simple puisque les deux seuls arguments utilisés ici sont le nom de la fonction ainsi que le nombre de cœurs[11]. Il conviendra d'exécuter cette fonction après avoir démarré le mode parallèle ("sfInit()") et de clôturer la session parallèle après l'appel de la fonction ("sfStop"). Le code ci-dessous est donc extrêmement simple, mais d'une puissance inouïe.

[11] Ici, je n'utilise que 4 cœurs pour ne pas solliciter l'ensemble des cœurs de ma machine.

```
#Chargement du package R
library("snowfall")

setwd("D:/ANR-EPI/")

FichList <-c("D151E", "D152Z", "D153E", "D158K", "D158V",
             "D159F", "D159G", "D151A", "D151C", "D155A",
             "D155C", "D156A", "D157A", "D158A")

# Définition du nombre de cœurs
 nb.cpus=4

#Initialisation de la parallélisation avec le nombre de cœurs choisis

sfInit(parallel=TRUE, cpus=nb.cpus)

# Appel de la fonction sfClusterApplyLB avec comme arguments la liste FichList des
#valeurs à appliquer à la fonction Sweave.paral

sfClusterApplyLB(FichList, Sweave.paral)

 # arret de la parallélisation
 sfStop()
```

C'est toute la puissance de R et la simplicité de Sweave qui est utilisée dans cette approche. On remarquera la fonction sfClusterApplyLB() est similaire dans son fonctionnement et dans sa syntaxe aux fonctions apply() et lapply(), bien connues des utilisateurs de R. Bien entendu, on peut utiliser ces quelques lignes de commande pour d'autres usages que Sweave puisque la fonction "sfClusterApplyLB()" permet d'envoyer des *process* sur différents cœurs dès lors qu'il n'y a pas (ou peu) d'interactions entre ces *process*. Le gain en temps est dans mon cas très appréciable, à tel point que j'ai voulu quantifier ces gains en fonction du nombre de cœurs alloués au parallélisme.

4. Banc test

Je propose ici les résultats d'un banc test minimal destiné à mesurer les gains en performance de la parallélisation dans notre cadre particulier. Je présente ici les résultats réalisés sur le fichier Sweave proposant une batterie d'estimations non-paramétriques de frontières de production sur un serveur de calcul de notre unité[12]. Ces estimations se basent sur les données de firmes des différents secteurs agro-alimentaires. Il est à noter que le nombre d'observations pour chacun des fichiers de données utilisés est variable.

[12]Il s'agit d'un serveur de 80 cœurs basé sur une architecture Xeon E7-4870 avec 40 bi-cœurs @2.40 Ghz, équipé avec 292 Go de RAM. Une belle bête !

(5a) Sur un programme simplifié

(5b) Sur le programme original avec 399 bootstraps

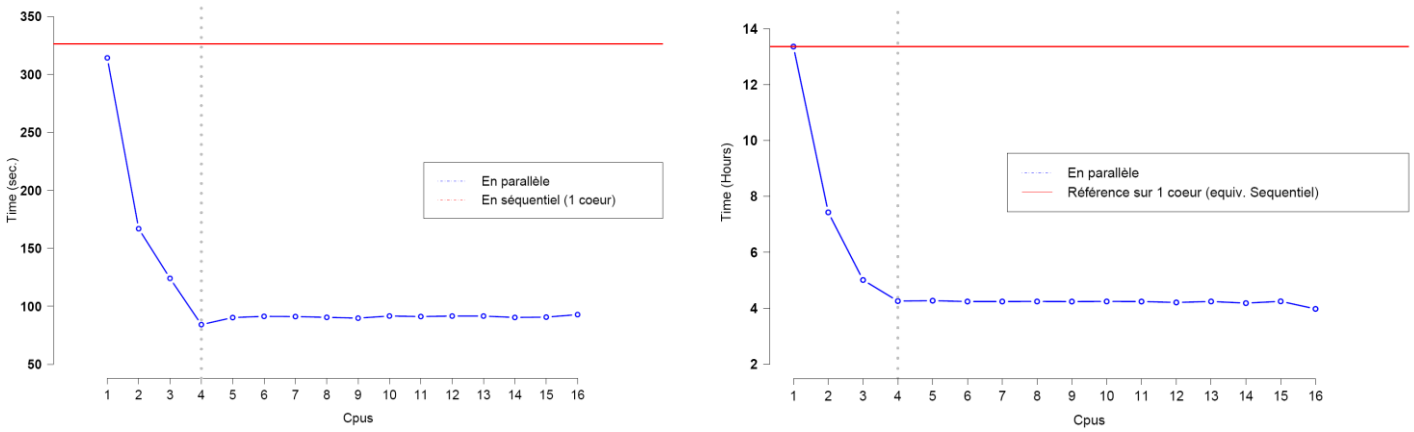


Figure 5. *Temps de calcul pour la production des 14 secteurs avec Sweave en séquentiel ou en parallèle.*

Dans la **Figure 5a**, on reporte en ordonnée, le temps nécessaire à la production des 14 rapports issus d'une version simplifiée du programme Sweave *MonAnalyse.Rnw* pour les 14 secteurs sans utilisation de tests (et donc sans bootstraps) pour, en abscisse, un nombre croissant de cœurs. Le même travail est également effectué avec ce même programme sur les 14 fichiers de données en utilisant la version séquentielle. Afin de faciliter la comparaison, on a tracé (en rouge) une droite correspondant au temps de calcul du programme séquentiel, lequel est légèrement supérieur au temps de calcul avec notre procédure parallèle sur un seul cœur (sans parallélisation donc). Il y a donc un gain à utiliser le programme parallèle sur un seul cœur plutôt que le programme séquentiel. Ceci est probablement dû à l'utilisation d'une fonction R dans le programme parallèle contrairement au programme séquentiel. On remarque que le temps de calcul décroît fortement à mesure que l'on augmente le nombre de cœurs. Avec 4 cœurs on a réduit le temps de calcul par rapport à la procédure séquentielle d'un rapport de 3,60, ce qui est déjà extrêmement intéressant. Le temps de calcul stagne[13] ensuite à cette valeur même en ajoutant de nombreux cœurs et semble alors ne plus profiter des bénéfices du parallélisme. Ceci n'est surprenant qu'au premier abord et s'explique assez facilement : tout d'abord, les 14 fichiers de données sont fortement hétérogènes en nombre d'observations. Les mêmes calculs (ceux de notre programme *MonAnalyse.Rnw*) prennent donc plus ou moins de temps suivant le fichier de données utilisé. Ensuite, nous avons utilisé une procédure qui équilibre la charge de travail des cœurs (*LB* = *Load Balance*) de sorte qu'un cœur ayant fini sa tâche va en effectuer une autre sans attendre. Ces deux éléments nous permettent de comprendre ce palier. Le temps minimal observé reste le même que l'on utilise 4, 5 ou 14 cœurs[14], il est clair qu'au moins un cœur continue une tâche longue (sur un fichier très gros) tandis que les autres cœurs se partagent les tâches de la production des autres rapports.

La **Figure 5b** reporte les temps d'exécution pour la production des 14 rapports comprenant des estimations plus complexes utilisant des tests basés sur des tirages de 399 échantillons bootstrappés, donc demandant plus de

[13] La légère augmentation du temps de calcul entre 4 et 5 cœurs, ainsi que les légères fluctuations autour d'une valeur constantes sont peut-être liées à l'utilisation du serveur par d'autres utilisateurs pendant le test. Il est possible également que l'utilisation de 4 cœurs soit optimale dans le partage des 14 fichiers de taille différentes.

[14] Par curiosité, nous avons poussé le test jusqu'à 16 cœurs, même si ce nombre de cœurs dépassait le nombre de fichiers de données utilisés.

temps de calcul. Là encore, et à titre de référence, la droite rouge correspondant au temps de calcul avec un seul cœur, correspondant à ce qu'aurait donné le programme séquentiel[15]. On peut remarquer que les gains en performance sont assez similaires (gain d'un facteur 3,14 avec 4 cœurs) que ceux obtenus lors de l'utilisation du programme simplifié moins gourmand en calcul et que l'on retrouve le même plateau, probablement pour les mêmes raisons[16]. La version séquentielle n'ayant pas été exécutée, on pourra remarquer tout de même que l'utilisation d'un seul cœur, prend 48099,57 secondes soit plus de 13 h qui sont ramenées à un peu plus de 4 h avec 4 cœurs.

Il serait intéressant de poursuivre ces essais en utilisant des fichiers de même taille afin de vérifier les intuitions données sur l'apparition de ce plateau, l'hypothèse étant que ce plateau disparaîtrait et que les temps d'exécutions se réduiraient encore à mesure que le nombre de cœur augmenterait. Toutefois, cet exercice est malgré tout assez coûteux en temps et dépasse le message de ce papier. Nous cherchons avant tout ici à proposer une méthode simple pour l'utilisation de Sweave en parallèle et en montrer l'utilité. J'ai illustré les gains en me servant ici de machines à plusieurs cœurs et les gains en temps d'exécution sont très importants même avec peu de cœurs. Ces deux éléments montrent que l'on a aussi intérêt à mettre en œuvre cette procédure sur une machine de bureau "normale" possédant au moins 2 cœurs, ce qui est courant à l'heure actuelle.

Pour conclure…

Cette expérience d'exécution parallèle d'un programme Sweave afin de créer simultanément plusieurs rapports d'une même structure et comprenant les mêmes objets de recherche mais sur des fichiers différents, s'est avérée extrêmement positive. Outre l'opportunité de jouer avec des outils modernes et de résoudre mes problèmes, ce projet montre qu'il est désormais facile de mettre en œuvre des outils de parallélisme réputés complexes et inaccessibles. Il est en réalité réellement simple d'exécuter des tâches en parallèle puisque les outils sont bien au point maintenant et qu'il existe de nombreuses notices et retours d'expérience. C'est en particulier le cas pour la création de rapports de recherche (mais également des tableaux de bord, des synthèses, etc...) avec Sweave, mais cela fonctionne également très bien avec beaucoup d'autres fonctions, du moment qu'elles peuvent s'écrire sous R et qu'il n'y a pas (ou pas trop) d'interdépendance dans leur exécution[17]. Au final, j'ai réussi à produire 14 rapports différents en faisant fonctionner les cœurs de mon ordinateur plutôt que ma pauvre tête... et cela ne m'a pas pris très longtemps pour arriver à cela.

Remerciements

Je tiens à remercier très chaleureusement mes collègues Thibault Laurent (TSE-CNRS) et Olivier de Mouzon (TSE-INRA) pour leur précieuse aide dans ce travail. C'est de discussions dans leur bureau qu'est né ce projet et c'est par le partage de leur immense savoir qu'a été élaboré ce projet. Je tiens à remercier également Valérie Orozco (TSE-INRA) qui m'aide depuis des années à promouvoir une démarche de recherche reproductible dans les travaux que nous menons ; démarche que j'applique (en partie) ici.

---

[15]Programme séquentiel que je n'ai pas eu le courage, et surtout la patience, d'exécuter !

[16]La légère baisse du temps de calcul pour 16 cœurs peut sembler étrange et m'a intrigué. Comme il n'est pas logique que l'on ait le moindre gain à utiliser plus de cœurs que l'on a de secteurs, l'explication la plus plausible est celle d'une baisse de l'utilisation du serveur (qui est partagé entre différents utilisateurs de mon unité) entraînant un gain ponctuel de performance sur le cœur occupé à traiter le fichier le plus gros.

[17] Il existe des solutions plus complexes pour gérer le parallélisme dans des situations où l'on a des dépendances entre les différents calculs, mais ces solutions sont plus complexes à mettre en œuvre et dépassent le cadre de cet article.

Références bibliographiques

Bontemps C, Orozco V (2013) Des outils pour la recherche reproductible : Sweave et Statweave, *45è Journées de la Statistique* ; Toulouse, France.
URL: http://jds2013.sfds.asso.fr/wp-content/uploads/2013/05/Programme.pdf

Koenker R, Zeileis A (2009) On reproducible econometric research, *J Appli Econom* **24**(5), 833-847.
URL: http://www.jstor.org/stable/25608762

Lamport L (1995) LaTeX : A Document Preparation System: User's Guide and Reference Manual Pearson Education.

Laurent T (2013) Utiliser un makefille pour produire un document LaTeX, Rencontre des ingénieurs statisticiens toulousains.
http://www-gremaq.univ-tlse1.fr/perso/laurent/presentation%20inge_stat/presentation%2018-02-13/presentation%20Makefile.pdf

Leisch F (2002) Sweave : Dynamic generation of statistical reports using literate data analysis. in *W. Hardle and B. Ranz (eds), Compstat, Physica-Verlag HD*, 575-580.
URL: http://dx.doi.org/10.1007/978-3-642-57489-4_89

Meredith E, Racine J S (2009) Towards reproducible econometric research : The sweave framework. *J Appli Econom* **24**(2) 366-374.
URL: http://www.jstor.org/stable/40206278

Nature E (2013) Reducing our irreproducibility, *Nature* **496**, p. 398.
URL:
http://www.nature.com/polopoly_fs/1.12852!/menu/main/topColumns/topLeftColumn/pdf/496398a.pdf

Porzelius C, Knaus H BJ, Schwarzer G (2009) Easier Parallel Computing in R with snowfall and sfCluster. *The R Journal* **1**(1), 54-59.
URL: http://journal.r-project.org/archive/2009-1/RJournal_2009-1_Knaus+et+al.pdf

Racine J S J (2012) Rstudio: A platform-independent ide for R and sweave, J *Appl Econom* **27**,167-172.URL: http://www.jstor.org/stable/41337225