Traitement de données multi-spectrales par calcul intensif et applications chez l'homme en imagerie par résonnance magnétique nucléaire

Mélodie Angeletti

École doctorale n° 65 : Science de la Vie, Santé, Agronomie, Environnement

THÈSE

pour obtenir le grade de docteur délivré par

l'Université Clermont-Auvergne

Spécialité doctorale "Bio-Informatique"

présentée et soutenue publiquement par

Mélodie ANGELETTI

le 21 février 2019

Traitement de données multi-spectrales par calcul intensif et applications chez l'homme en imagerie par résonance magnétique nucléaire

Financée par l'allocation SANTE 2014 du Conseil régional d'Auvergne (http://www.auvergne.fr/)

Directeurs de thèse : **Jean-Marie BONNY et Jonas KOKO**

Jury

Mme. Camille Coti,	Maître de conférence, Université Paris XIII	Examinateur
M. Raphaël Couturier,	Professeur,Université de Montbéliard	Rapporteur
M. Franck Durif,	Professeur, Université Clermont Auvergne	Président du jury
Mme Évelyne Lutton,	Directrice de recherche, INRA-Agro Paris Tech	Rapporteur
Mme Hélène Toussaint,	Ingénieur de recherche, Université Clermont Auvergne	Examinateur
Mme Charlotte Sinding,	Chargée de recherche, INRA de Dijon	Examinateur

Laboratoire d'Informatique, de Modélisation et d'Optimisation des Systèmes (LIMOS),
CNRS UMR 6158, Aubière, France
INRA, AgroResonance - UR370 QuaPA, Centre Auvergne-Rhône-Alpes,
Saint-Gênes-Champanelles, France

THESE

Remerciements

Cette thèse n'aurait pas pu se faire sans l'aide de plusieurs personnes que je tiens à remercier. Pour commencer, je remercie la région Auvergne d'avoir financé ce sujet de thèse. Merci également à mon beau-frère et à ma sœur de m'avoir informé de l'existence de ce sujet de thèse. Je veux également exprimer toute ma gratitute à mes deux directeurs de thèse Jean-Marie Bonny et Jonas Koko. Je dois au premier une meilleure compréhension de l'imagerie par résonance magnétique et des mécanismes cérébraux mis en jeux lors de la prise alimentaire ainsi qu'une aide précieuse pour analyser les données d'imagerie. Mon second directeur m'a apporté son expertise pour le calcul parallèle en particulier sur les architectures à mémoire distribuée et m'a trouvé une formation sur la programmation des GPUs. Il m'a également fait suivre plusieurs offres d'emploi. Cette thèse étant pluridisciplinaire, je les remercie d'avoir suivi mon travail même lorsque leur domaine de compétence n'était pas concerné, et d'avoir relu mon manuscrit de thèse.

Mes remerciements vont ensuite à mes deux rapporteurs Mme Évelyne Lutton et M. Raphaël Couturier qui m'ont fait l'honneur de lire mon manuscrit de thèse malgré les 280 pages. Merci pour vos commentaire et vos rapports favorables. Je remercie également mes quatre examinateurs M. Franck Durif, Mme Camille Coti, Mme Hélène Toussaint et Mme Charlotte Sinding pour avoir accepté de participer à mon jury de thèse. J'adresse une mention spéciale à Charlotte qui m'a suivie depuis le début de ma thèse en participant à mes comités de thèse.

Je tiens également à remercier les doctorants avec qui j'ai pu échanger tout au long de cette thèse. J'adresse une mention spéciale à mes co-bureaux de la salle F203 : Nina, Matthieu, Kergann, Antoine, Giaccomo et Nestor . J'ai passé trois années (et demi même) formidables avec vous notamment lors des goûters d'anniversaires.

Enfin, je remercie ma famille et ma (future) belle-famille pour m'avoir soutenu pendant les moments difficiles. Je suis également infiniment reconnaissante envers mon chéri pour son soutien inconditionnel. Il m'a épaulé pendant les moments de doute, a accepté de me faire répéter chacune de mes présentations et il s'est même dévoué pour relire le manuscrit.

Un grand merci, sans toi, je ne serai (peut-être) pas allée au bout de cette thèse.

Résumé

L'imagerie par résonance magnétique fonctionnelle (IRMf) étant une technique non invasive pour l'étude de cerveau, elle a été employée pour comprendre les mécanismes cérébraux sous-jacents à la prise alimentaire. Cependant, l'utilisation de stimuli liquides pour simuler la prise alimentaire engendre des difficultés supplémentaires par rapport aux stimulations visuelles habituellement mises en œuvre en IRMf. L'objectif de cette thèse a donc été de proposer une méthode robuste d'analyse des données tenant compte de la spécificité d'une stimulation alimentaire. Pour prendre en compte le mouvement dû à la déglutition, nous proposons une méthode de censure fondée uniquement sur le signal mesuré. Nous avons de plus perfectionné l'étape de normalisation des données afin de réduire la perte de signal.

La principale contribution de cette thèse est d'implémenter l'algorithme de Ward de sorte que parcelliser l'ensemble du cerveau soit réalisable en quelques heures et sans avoir à réduire les données au préalable. Comme le calcul de la distance euclidienne entre toutes les paires de signaux des voxels représente une part importante de l'algorithme de Ward, nous proposons un algorithme « cache-aware » du calcul de la distance ainsi que trois parallélisations sur les architectures suivantes : architecture à mémoire partagée, architecture à mémoire distribuée et GPU NVIDIA.

Une fois l'algorithme de Ward exécuté, il est possible d'explorer toutes les échelles de parcellisation. Nous considérons plusieurs critères pour évaluer la qualité de la parcellisation à une échelle donnée. À une échelle donnée, nous proposons soit de calculer des cartes de connectivités entre les parcelles, soit d'identifier les parcelles répondant à la stimulation à l'aide du coefficient de corrélation de Pearson.

Mots-clés : IRMf alimentaire, Parcellisation multi-échelle, Algorithme de Ward, Distance euclidienne, Parallélisation, OpenMP, MPI, CUDA

Abstract

As a non-invasive technology for studying brain imaging, functional magnetic resonance imaging (fMRI) has been employed to understand the brain underlying mechanisms of food intake. Using liquid stimuli to fake food intake adds difficulties which are not present in fMRI studies with visual stimuli. This PhD thesis aims to propose a robust method to analyse food stimulated fMRI data. To correct the data from swallowing movements, we have proposed to censure the data uniquely from the measured signal. We have also improved the normalization step of data between subjects to reduce signal loss.

The main contribution of this thesis is the implementation of Ward's algorithm without data reduction. Thus, clustering the whole brain in several hours is now feasible. Because Euclidean distance computation is the main part of Ward algorithm, we have developed a cache-aware algorithm to compute the distance between each pair of voxels. Then, we have parallelized this algorithm for three architectures: shared-memory architecture, distributed memory architecture and NVIDIA GPGPU.

Once Ward's algorithm has been applied, it is possible to explore multi-scale clustering of data. Several criteria are considered in order to evaluate the quality of clusters. For a given number of clusters, we have proposed to compute functional connectivity maps between clusters or to compute Pearson correlation coefficient to identify brain regions activated by the stimulation.

Keywords: food fMRI, multi-scale clustering, Ward's algorithm, Euclidean distance, Parallelisation, OpenMP, MPI, CUDA

Table des matières

Table des figures

Liste des tableaux

Liste des Algorithmes

Liste des symboles

Vecteurs et Matrices

$\boldsymbol{x}$	un vecteur
$\boldsymbol{A}$	une matrice
$\boldsymbol{A}^\top$	la transposée de la matrice $\boldsymbol{A}$
n_V	nombre d'éléments ou nombre de voxels en IRMf
n_T	nombre de caractéristiques ou nombre de points temporels en IRMf
$\boldsymbol{X}$	matrice de données de taille $n_\mathrm{T} \times n_\mathrm{V}$
$\boldsymbol{x}_i$	la colonne i de $\boldsymbol{X}$: $\boldsymbol{X} = [\boldsymbol{x}_1 \dots \boldsymbol{x}_{n_\mathrm{V}}]$.
x_{ij}	élément de la matrice $\boldsymbol{X}$ à la ligne i et à la colonne j
$\boldsymbol{D}$	matrice de distances de taille $n_\mathrm{V} \times n_\mathrm{V}$
n_{CB}	le nombre de tuiles nécessaire pour couvrir toutes les colonnes de la matrice $\boldsymbol{X}$
n_{RB}	nombre de blocs nécessaire pour couvrir les lignes de la matrice $\boldsymbol{X}$
m_C	le nombre de colonnes par tuile
m_R	nombre de lignes par tuile
${m_C}^*$	nombre de colonnes donnant les performances optimales
${m_R}^*$	nombre de lignes donnant les performances optimales
$\mathcal{I}_i$	le $i^{ème}$ bloc de lignes
$\mathcal{J}_j$	la $j^{ème}$ tuile de colonnes

$$\mathcal{J}_j = \begin{cases} [1+(j-1)m_C, \dots, j \times m_C] \\ [1+(j-1)m_C, \dots, n_\mathrm{T}] \text{ si } j = n_{CB} \end{cases} \qquad \mathcal{I}_i = \begin{cases} [1+(i-1)m_R, \dots, i \times m_R] \\ [1+(i-1)m_R, \dots, n_\mathrm{T}] \text{ si } i = n_{RB} \end{cases}$$

$\boldsymbol{X}_{i\cdot}$	la sous-matrice de $\boldsymbol{X}$ couvrant les lignes du bloc $\mathcal{I}_i$ de sorte que

$$\boldsymbol{X} = [\boldsymbol{X}_{1\cdot}; \boldsymbol{X}_{2\cdot}; \dots; \boldsymbol{X}_{n_{RB}\cdot}]^\top$$

$\boldsymbol{X}_{\cdot j}$	la sous-matrice de $\boldsymbol{X}$ couvrant les colonnes de la tuile $\mathcal{J}_i$ de sorte que

$$\boldsymbol{X} = [\boldsymbol{X}_{\cdot 1}\boldsymbol{X}_{\cdot 2}, \dots, \boldsymbol{X}_{\cdot n_{CB}}]$$

$\boldsymbol{X}_{IJ}$	la sous-matrice de $\boldsymbol{X}$ couvrant les lignes du bloc $\mathcal{I}_I$ et les colonnes de la tuile $\mathcal{J}_J$
$\mathbf{1}_{\boldsymbol{n}}$	le vecteur unitaire de taille n
$\boldsymbol{D}_{KL}$	la distance entre les colonnes de la tuile $\cdot K$ et les colonnes de la tuile $\cdot L$

$\sum^{col}$ l'opération de sommation des lignes d'une matrice $\boldsymbol{X}$ colonne par colonne définie par

$$\sum^{col} \boldsymbol{X} = \left(\sum_{i=1}^{n_\mathrm{T}} \boldsymbol{x}_{i1},\ \ldots,\ \sum_{i=1}^{n_\mathrm{T}} \boldsymbol{x}_{in_\mathrm{V}} \right) \quad .$$

$\|\boldsymbol{x}\|_2$ norme euclidienne du vecteur $\boldsymbol{x}$ définie par $\|\boldsymbol{x}\|_2 = \sqrt{\sum_{i=1}^{n_\mathrm{T}} x_i}$

$\boldsymbol{s}$ le vecteur des normes des colonnes défini par $\boldsymbol{s} = \left(\|\boldsymbol{x}_1\|_2^2, \ldots, \|\boldsymbol{x}_{n_\mathrm{V}}\|_2^2\right) = \left(\sum_{t=1}^{n_\mathrm{T}} x_{ti_1}, \ldots, \sum_{t=1}^{n_\mathrm{T}} x_{ti_{n_\mathrm{V}}}\right)$

$\boldsymbol{s_A}$ la restriction du vecteur $\boldsymbol{s}$ aux colonnes de la matrice $\boldsymbol{A}$

$\hat{}^p$ l'élévation de puissance p de la matrice élément par élément définie par

$$\boldsymbol{X}^{\hat{}p} = \begin{pmatrix} \boldsymbol{x}_{11}^p & \ldots & \boldsymbol{x}_{1n_\mathrm{V}}^p \\ \ldots & \ddots & \ldots \\ \boldsymbol{x}_{n_\mathrm{T}1}^p & \ldots & \boldsymbol{x}_{n_\mathrm{T}n_\mathrm{V}}^p \end{pmatrix}$$

Parcellisation

$C_j^{n_\mathrm{C}}$ la $j^{ème}$ parcelle pour un découpage en n_C parcelles

$\left|C_j^k\right|$ cardinal de la parcelle C_j^k autrement dit son nombre d'éléments

$\overline{C_j^k}$ le centroïde de la $j^{ème}$ parcelle pour un découpage en n_C parcelles défini par :

$$\overline{C_i^k} = \frac{1}{\left|C_i^k\right|} \sum_{\boldsymbol{x}_i \in C_i^k} \boldsymbol{x}_i \quad = \begin{pmatrix} \frac{1}{\left|C_i^k\right|} \sum\limits_{\boldsymbol{x}_i \in C_i^k} x_{1i} \\ \vdots \\ \frac{1}{\left|C_i^k\right|} \sum\limits_{\boldsymbol{x}_i \in C_i^k} x_{n_\mathrm{T}i} \end{pmatrix}$$

$\mathrm{d}(C_i^k, C_j^k)$ dissimilarité ou « distance » entre les parcelles

$\delta(\boldsymbol{x}_i, \boldsymbol{x}_j)$ métrique utilisée pour calculer la distance entre deux vecteurs

n_C nombre de parcelles choisi

n_C^* nombre optimal de parcelles

Paramètres de mouvement

$\Delta x(t)$ Translation (**cf. Figure 1.2**)

$\Delta y(t)$ Translation (**cf. Figure 1.2**)

$\Delta z(t)$ Translation (**cf. Figure 1.2**)

$roll(t)$ Rotation (**cf. Figure 1.2**)

$pitch(t)$ Rotation (**cf. Figure 1.2**)

$yaw(t)$ Rotation (**cf. Figure 1.2**)

$TAD(t)$ *Translational Absolute displacement* défini par $|\Delta x(t)| + |y(t)| + |\Delta z(t)|$

$RAD(t)$ *Rotational Absolute displacement* défini par $|roll(t)| + |pitch(t)| + |yaw(t)|$

$x'(t)$ Dérivée de x (t)

$y'(t)$ Dérivée de y (t)

$z'(t)$ Dérivée de z (t)

$roll'(t)$ Dérivée de roll (t)

$pitch'(t)$ Dérivée de pitch (t)

$yaw'(t)$ Dérivée de yaw (t)

FD (t) *Framewise displacement* défini par $FD(t) = |x'(t)|+|y'(t)|+|z'(t)|+|roll'(t)|+|pitch'(t)|+|yaw'(t)|$

TAD' (t) $TAD'(t) = |x'(t)| + |y'(t)| + |z'(t)|$

RAD' (t) $RAD'(t) = |roll'(t)| + |pitch'(t)| + |yaw'(t)|$

Acronymes

A

ACI *Analyse en composantes indépendantes* 18–24, 29

ACP *Analyse en composantes principales* 18, 22, 29

B

BOLD *Blood oxygenation level dependent* 3, 4, 6, 7, 9, 12, 17, 49

C

CPU *Central Processor Unit* 32, 37, 39, 40, 45–47, 103, 125, 128, 129, 136, 143, 144, 148, 205, 206

E

EPI *Image écho planaire* 3, 15, 56–58, 78

F

FDR *False Discovery Rate* 8, 16

FWER *Familywise Error Rate* 7, 16

G

GEMM *General Matrix Multiplication* 124, 128

GLM *Modèle linéaire général* 11, 13, 15–17, 21, 27–30, 77, 182, 192, 207

GPU *Graphical Processor Unit* . 37–41, 45–47, 51, 52, 103, 122, 124, 125, 127–129, 136, 140, 143–145, 148, 150, 165, 199, 201, 202, 205

H

HAC *Clustering hiérarchique agglomératif* 26, 27, 29, 30

HRF *Réponse hémodynamique* 12, 15, 17

I

IRM *Imagerie par résonance magnétique* 3

IRMf *Imagerie par résonance magnétique fonctionnelle* 3, 4, 6–13, 15–18, 20, 21, 23, 25–27, 29, 49–51, 55, 73, 93, 102, 103, 199, 207

ISC *Corrélation inter-sujet* 24–26, 29

M

MVPA *Multivariate pattern analysis* 13, 207

P

PSC *Percent signal change* 77, 78, 174

R

ROI *Région d'intérêt* 185

S

SM *Streaming Multiprocessor* 40, 46

SVM *Support Vector Machine* 14

T

TCA *Temporal clustering analysis* 28, 29

TE *Temps d'écho* 4, 56

TR *Temps de répétition* 4, 56

Première partie

Introduction

Chapitre 1

Introduction à l'imagerie par résonance magnétique fonctionnelle

L'imagerie par résonance magnétique fonctionnelle (IRMf) est l'une des techniques non-invasives les plus employées chez l'homme pour l'étude des processus cérébraux avec une bonne résolution anatomique. L'intérêt de l'IRMf est d'étudier la connectivité cérébrale et la coopération des aires cérébrales pour des processus complexes [9]. Elle permet la localisation des réseaux neuronaux activés, de connaître l'amplitude de l'activité et la dynamique des processus cérébraux. On distingue deux sortes d'expérience : la première où les sujets doivent effectuer une tâche plus ou moins complexe et la deuxième où les sujets sont au repos. Pendant longtemps, l'activité au repos a été considérée comme une activité accessoire dont il faut tenir compte pour l'analyse des images. Récemment, elle est devenue l'objet de recherches spécifiques [174]. En effet l'activité au repos est fondée sur la connectivité des aires cérébrales et cette connectivité peut être altérée par des processus dégénératifs comme le vieillissement [138] ou les démences [192]. Dans cette thèse, cet aspect ne sera pas traité puisque l'intérêt porte sur les processus cérébraux momentanés induits par des stimulations alimentaires.

1.1 Les principes et objectifs de l'IRMf de type BOLD

1.1.1 Imagerie par résonance magnétique nucléaire

Ce paragraphe n'a pas pour objet de décrire en détail l'imagerie par résonance magnétique (IRM) mais d'en donner les grands principes. Cette technique d'imagerie repose sur la notion de spin qui est une propriété physique des noyaux au même titre que la masse. La résonance magnétique nucléaire repose sur cette propriété propre à certains noyaux atomiques possédant ce spin. Lorsque plusieurs noyaux sont placés dans un champ magnétique externe, ils absorbent l'énergie du rayonnement radiofréquence à une fréquence précise, d'où le terme de résonance. L'IRM nécessite un champ magnétique intense, homogène et stable afin de permettre l'expérience au sein des tissus biologiques. À l'aide d'antennes placées à l'intérieur de l'aimant et à proximité des tissus d'intérêt, des champs radiofréquences sont appliqués sous forme d'impulsions afin de mesurer un signal qui rend compte d'un grand nombre de spins. La spécificité de l'IRMf est de pouvoir localiser précisément l'origine de ce signal, et de reconstruire ainsi une image en trois dimensions du cerveau (par exemple). Pour des raisons de sensibilité, les protons de l'eau et des gras sont les noyaux principalement ciblés en IRM chez l'homme. L'imagerie écho-planaire (**EPI** pour *echo-planar imaging*) est la méthode d'acquisition la plus communément utilisée pour deux raisons principales ; elle est rapide (une image de l'ensemble du cerveau entre 500 ms et quelques secondes [68]) et elle est sensible au contraste *Blood oxygenation level dependent* (BOLD) sur lequel repose généralement l'IRMf, en particulier dans ce travail.

On appelle **temps de relaxation** le temps nécessaire pour que les protons retrouvent leur

état initial. Il existe deux types de temps de relaxation : **le T1 longitudinal** (c'est-à-dire parallèle au champ magnétique imposé) et **le T2 perpendiculaire**. L'intensité du signal dépend du temps de relaxation et de la densité des protons. Cependant, lors d'une expérience d'IRM, ce temps T2 décroît plus que ce qui est prédit par les mécanismes moléculaires. On appelle alors **T2*** le temps de relaxation transverse observé qui reflète à la fois le vrai temps de relaxation T2 et les inhomogénéités du champ magnétiques [198]. On appelle **temps de répétition (TR)** l'intervalle de temps entre deux impulsions magnétiques [9]. On appelle **temps d'écho (TE)** le temps écoulé entre l'impulsion magnétique et la détection du signal émis. Le contraste (image plus ou moins grise) dépend alors de la densité d'eau dans le tissu, de son interaction avec les macromolécules environnantes et des deux variables que sont le temps de répétition et le temps d'écho. On appelle images **pondérées en T1** les images acquises avec des temps de relaxation et des temps d'écho court. Les images dites **pondérées en T2** sont acquises avec un long temps de relaxation et un long temps d'écho. Les images pondérées en T1 fournissent un bon contraste entre la substance grise (en gris) et la substance blanche (en blanc) alors que le liquide cérébro-spinal ne contient pas de signal. Les images pondérées en T2 fournissent, quant à elles, un contraste entre le liquide cérébro-spinal et les tissus cérébraux [103].

1.1.2 Contraste BOLD

L'IRMf repose sur la théorie du couplage neurovasculaire décrite par Roy *et al.* [177] selon laquelle l'activité neuronale entraîne une augmentation régionale de la consommation en oxygène et donc une augmentation du débit sanguin cérébral local. La méthode de référence est l'utilisation du contraste BOLD introduit par Ogawa *et al.* [158]. Le contraste BOLD prend appui sur les propriétés magnétiques différentes entre l'hémoglobine dés-oxygénée et l'hémoglobine oxygénée. L'hémoglobine désoxygénée est en effet paramagnétique au contraire de l'hémoglobine oxygénée, qui elle est diamagnétique.L'activation d'une zone cérébrale entraîne une augmentation de la consommation en oxygène et donc une augmentation du débit et du volume sanguin cérébral, à proximité des neurones dont l'activité varie. Ces variations métaboliques donnent lieu au contraste BOLD. Il y a généralement un surcroît de concentration en sang oxygéné dans les vaisseaux proches des neurones actifs. Le ratio hémoglobine oxygénée sur hémoglobine désoxygénée noté Hb/HbO2 augmente, ce qui conduit à une augmentation du signal IRM (effet diamagnétique favorable au signal RMN) [58]. Il parait important d'insister sur le fait que l'IRMf de type BOLD ne se fonde pas sur la détection directe de l'activité électrique des neurones, mais sur un couplage neurovasculaire dont les déterminants sont multiples. Le contraste BOLD naît de l'interaction entre trois facteurs (consommation d'oxygène, débit et volume sanguin) dont les contributions respectives sont très variables spatialement, d'un sujet à un autre ou selon le type de stimulation. Beaucoup de modèles sont disponibles pour expliquer les signaux observés [195].

En général, le signal BOLD augmente environ deux secondes après le début de l'activité neuronale et atteint son maximum après environ six secondes. Tant que l'activité neuronale, continue, le signal BOLD reste à son maximum puis revient à sa ligne de base (sa valeur sans activité neuronale) au bout d'une dizaine de secondes après l'arrêt de l'activité neuronale [5].

1.1.3 Expérience en IRMf

Au cours d'une expérience d'IRMf, les sujets sont placés dans le champ magnétique d'un aimant horizontal et sont donc allongés sur le dos. Au cours d'une expérience d'IRMf fondée sur des tâches, les sujets sont soumis à des stimuli. Pour améliorer la détection des variations du signal, ces stimuli sont répétés au cours de l'expérience en alternance avec des périodes de repos afin d'améliorer la puissance statistique de l'expérience car les variations sont de faible amplitude. L'ordre des stimuli ainsi que leur durée est appelé paradigme de l'expérience. Une étude IRMf comporte trois étapes majeures (détaillée en Figure 1.1) :

1. la conception de l'étude (Experimental design **cf. Fig 1.1**)

2. l'acquisition des données, effectuée généralement sur une cohorte de sujets (Data acquisition et Reconstruction **cf. Fig 1.1**)
3. le traitement et l'analyse des données (Preprocessing et Data Analysis **cf. Fig 1.1**), étape cruciale car les données sont extrêmement bruitées.

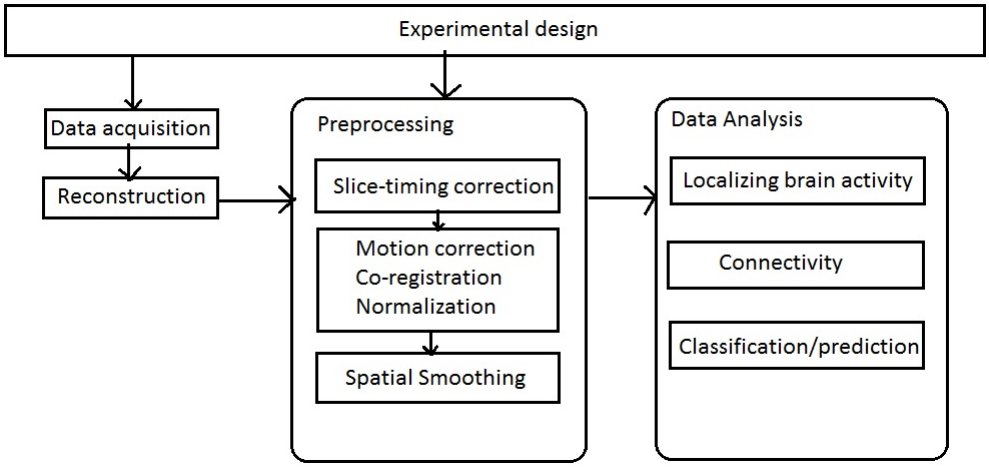


FIGURE 1.1 – Les étapes majeures d'une expérience d'IRMf issue de [127].

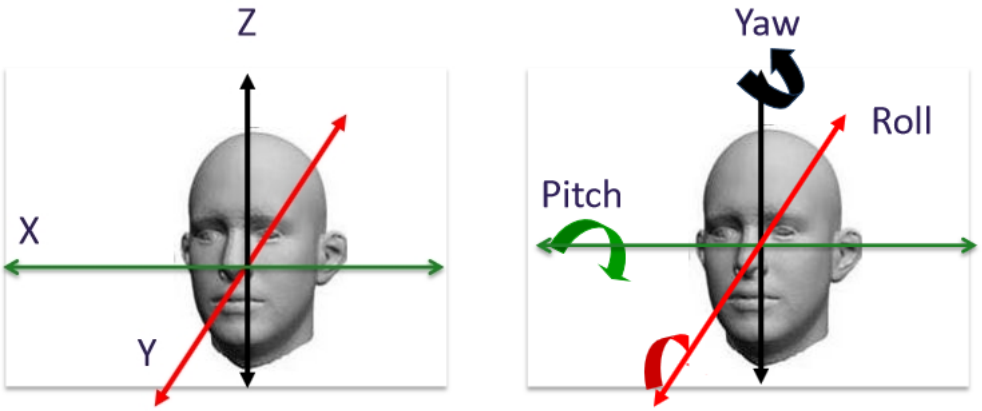
Lors de l'acquisition d'une image sur tout le cerveau, l'image est en fait constituée de plusieurs coupes du cerveau acquises séquentiellement, donc à des temps différents. L'étape de correction des coupes temporelles (*slice timing correction* **cf. Figure 1.1**) a pour but le réalignement temporel les coupes du cerveau afin que l'ensemble des coupes corresponde à un même instant d'acquisition. La deuxième étape (*Motion correction*) consiste à ré-aligner la série d'images temporelles afin de corriger le mouvement du sujet au cours de l'expérience. Le but est de limiter l'impact du mouvement sur les signaux temporels mesurés localement. Pour cela, six paramètres de mouvement pour chaque image sont estimés par rapport à une image de référence (la première image ou l'image moyenne par exemple). Ces six paramètres de mouvement sont composés de trois translations (en mm) et de trois rotations (en degré). La Figure 1.2 montre les six paramètres de réalignement ou de recalage rigide. La *coregistration* sert à aligner les images fonctionnelles sur les images structurelles obtenues sur le même sujet pendant le même expérience. La normalisation vise à placer chaque image de cerveau dans un repère neuroanatomique de référence par une transformation non-linéaire afin de combiner les images de plusieurs sujets. Dans ce repère, une région cérébrale peut-être identifiée grâce à ses coordonnées. Cette étape gomme la variabilité inter-sujets et permet d'identifier les zones du cerveau activées. Enfin la dernière étape du prétraitement, le lissage spatial, consiste à convoluer les images fonctionnelles avec un noyau gaussien afin de réduire le bruit aléatoire et de faire en sorte que le bruit se rapproche d'une distribution gaussienne, prérequis aux inférences statistiques faites en fin de traitement [127]. En réduisant la résolution spatiale, il permet d'améliorer le recouvrement spatial des acquisitions provenant de différents sujets qui proviennent d'une normalisation imparfaite et ainsi d'améliorer les inférences de groupe .

Une fois les prétraitements terminés, plusieurs méthodes d'analyse sont envisageables afin de déterminer quelles régions cérébrales sont activées par la stimulation.

1.2 Les principales difficultés

1.2.1 Inconvénients du contraste BOLD

La section 1.1.2 montre que le signal BOLD est une mesure indirecte de l'activité neuronale. De plus, le signal BOLD est transitoire (une dizaine de secondes environ) et correspond à une



(a) Translations (b) Rotations

FIGURE 1.2 – Paramètres de recalage rigide issus de l'étape de réalignement.

variation de signal par rapport à l'état de repos de l'ordre du pourcent voire moins [118, 209], le rendant difficile à détecter à cause du bruit (**cf. section 1.2.3**). Si l'IRMf bénéficie d'une résolution spatiale meilleure que les autres techniques d'imagerie cérébrale, à cause du retard de la réponse hémodynamique au changement métabolique, sa résolution temporelle est faible, si on la compare à la durée des signaux électriques qui la sous-tendent. À cause de la distorsion spatiale ou de la perte de signal dans les régions orbito-frontales et pariétales latérales (à cause de la différence de susceptibilité magnétique entre l'air et les tissus cérébraux,**cf. Section A.2** pour une localisation), le signal BOLD peut ne pas être détecté dans les régions ventrales et temporales et dans le cortex préfrontal [76], donnant alors des faux-négatifs. De plus, le signal BOLD ne peut pas facilement différencier un processus fonctionnel spécifique d'une neuromodulation (qui est la régulation d'un groupe de neurones par un autre neurone) [132]. De plus, la réponse hémodynamique est influencée par le nombre de neurones activés, nombre de neurones qui peut changer spatialement ou temporellement. Dans les régions corticales où peu de groupes de neurones répondent à la stimulation, la diffusion des neuro-transmetteurs peut dominer la réponse hémodynamique et rendre impossible la détermination du rôle exact de l'aire cérébrale [132]. L'intensité du signal BOLD est limitée en résolution spatiale du fait de l'étalement de la perfusion au-delà du strict territoire des neurones activés. Ceci est généralement contrecarré - dans une certaine limite - par l'augmentation du champ statique [210].

1.2.2 Imperfections de l'imagerie

À champ inférieur ou égal à 3 T, l'imagerie rapide utilisée en IRMf est sensible aux variations du T2*, temps de relaxation spin-spin apparent. Ces approches sont sensibles aux inhomogénéités du champ magnétique macroscopique qui proviennent par exemple des différences de susceptibilité magnétique entre les substances présentes dans la boîte crânienne. Il en résulte des distorsions géométriques [108], le plus souvent non-rigides, et des modulations du signal évoquées dans le paragraphe précédent [222]. En résumé, les risques liés aux imperfections de l'imagerie sont essentiellement des erreurs de localisation des territoires activés et une mauvaise détection (faux négatif) dans les territoires proches des interfaces

1.2.3 Bruits

Les données temporelles d'IRMf acquises contiennent à la fois du signal BOLD lié à l'activité cérébrale et des signaux de bruit dont l'amplitude est souvent du même ordre de grandeur voire plus grande que l'amplitude du signal d'intérêt. Avec l'émergence de l'IRMf au repos qui n'utilise

pas de stimulation, mais dont les résultats sont très sensibles aux mouvements spontanés du sujet, l'analyse du bruit et en particulier du mouvement a connu récemment un regain d'intérêt.

Liu a proposé récemment une revue des différentes contributions du bruit au signal d'IRMf [129] que nous reprenons ici. La première source de bruit est le bruit de fond composé du bruit thermique et du bruit radio-fréquence. Le bruit thermique résulte de l'ensemble des fluctuations aléatoires induites par l'agitation thermique le long de la chaîne électronique de détection du signal IRM (antennes, amplificateurs, convertisseurs, ...). Le bruit thermique est inhérent à l'acquisition et ne peut donc pas être évité. En revanche, il peut être réduit en filtrant tout signal en dehors de la bande de fréquence d'intérêt. Les bruits radio-fréquence peuvent être dus au contact intermittent de deux pièces métalliques ou à l'émergence soudaine d'un signal radio dans l'environnement du scanner. Une autre source de bruit est constituée par la dérive basse fréquence des signaux mesurés, partiellement dues aux instabilités du scanner. Elles sont problématiques lorsqu'elles sont dans la même bande de fréquence que la tâche réalisée. C'est pourquoi le paradigme de l'expérience en IRMf dure en général moins de soixante secondes.

La deuxième source principale est le bruit physiologique. Le bruit physiologique résulte des mouvements du sujet qu'ils soient périodiques, cardiaques et respiratoires, ou provoqués par les mouvements conscients du sujet. Leurs effets en IRMf sont complexes car ils induisent plusieurs perturbations simultanées ; un déplacement des tissus, des variations de la distribution du champ magnétique et des modulations métaboliques au cours du temps. Ces perturbations produisent le plus souvent des modulations non-linéaires du signal, à distance de la source (le champ magnétique dans le cerveau est modulé par la déglutition par exemple [25]) et pas toujours de manière synchrone (effet de la relaxation par exemple).

La troisième source principale de bruit est le mouvement. Si le mouvement du sujet décale les images de la largeur d'un voxel à partir de la moitié de l'expérience, le signal est alors acquis pour un voxel pour la première moitié du signal et pour un voxel voisin pour la seconde moitié de l'expérience. Dans ce cas le signal mesuré ne représente pas la réponse d'un seul et même voxel tout au long de l'expérience. Concrètement, le signal d'un voxel est composé des signaux des voxels voisins et cette combinaison change au cours du temps en fonction des mouvements effectués par le sujet. Les changements de signaux sont particulièrement importants pour des voxels à la frontière de plusieurs tissus cérébraux pour lesquels l'intensité du signal varie entre les tissus. La correction des artéfacts de mouvement est détaillée ci-dessous.

1.2.4 Reproductibilité et réplicabilité des résultats

Après que Ioannidis a montré que la majorité des résultats publiés sont faux [105], plusieurs articles se sont interrogés sur la validité des études en IRMf [27, 62]. Eklund *et al.* ont en effet montré que les études d'IRMf peuvent avoir des taux de faux positifs bien supérieurs au seuil des 5% choisi (qui correspond au nombre de voxels considérés comme actifs alors qu'ils sont inactifs). Ils ont trouvé que pour une analyse individuelle, l'analyse avec le logiciel SPM peut atteindre 70% de faux positifs pour des temps de répétitions courts et/ou un paradigme par blocs avec des blocs lents [61] en cause d'une mauvaise modélisation de la fonction d'auto-corrélation temporelle. Ils ont noté cependant, que le lissage spatial permet de réduire le taux de faux positifs. Pour l'analyse de groupe, ils ont montré que même en prenant d'autres logiciels d'analyse, le taux de faux positifs pour les tests paramétriques avec une correction FWER (**cf. section 2.1.5**) atteint également 70% dans certains cas lorsque sont considérés les tests à l'échelle des clusters (groupe de voxels) remettant ainsi à cause la validité de nombreuses études d'IRMf. Les tests à l'échelle du voxel sont conservatifs mais valides et en dessous du seuil des 5% [62]. Les tests non-paramétriques (à l'échelle des clusters et à l'échelle des voxels) produisent un taux de faux positifs en dessous de 5%. Ils attribuent ces taux de faux positifs élevés à une violation des hypothèses de la théorie des champs aléatoires gaussiens (*Random field theory*) : la première violation concerne sur la forme de l'auto-corrélation spatiale qui plutôt que d'être gaussienne et une queue plus longue et la deuxième réfute l'hypothèse d'une auto-corrélation spatiale constante

dans l'ensemble du cerveau [63]. Cox *et al.* ont réfuté ce taux alarmiste de 70% de faux positifs, obtenant sur les mêmes données qu'Eklund *et al.* [62] des taux de faux positifs allant jusqu'à 30% pour les tests paramétriques (les tests non-paramétriques étant en dessous des 5%) [49]. Kessler *et al.* ont nuancé les propos d'Eklund *et al.* en montrant que les clusters de voxels seuillés à 0,001 sont valides pour une correction FDR [116] pour les jeux de données initialement analysés par Eklund *et al.*. Cependant, Eklund *et al.* considèrent que les seuils donnés par Keller *et al.* ne sont pas facilement généralisables aux autres études déjà publiées [63].

Goodman *et al.* distinguent la réplicabilité de la reproductibilité [82]. Ils définissent la reproductibilité comme la capacité pour un chercheur de dupliquer les résultats d'une étude antérieure en utilisant les mêmes données et le même processus d'analyse [82]. La réplicabilité est définie comme la capacité à dupliquer les résultats d'une étude antérieure à partir d'une nouvelle expérimentation [82]. Poldrack *et al.* ont identifié plusieurs raisons expliquant la faible réplicabilité des résultats obtenus en IRMf [167]. La première raison est la faible puissance statistique des études à cause du nombre limité de sujets analysés et de la faible taille de l'effet mesuré, conjugués ces deux limitations permettent d'atteindre rarement la significativité. Dans ce contexte, Turner *et al.* suggèrent par exemple de constituer les groupes avec une centaine de sujets [208], ce qui est très rarement le cas. Une deuxième raison est la déficience des procédures de correction des tests multiples. Les tests statistiques étant répétés sur un grand nombre de voxels du cerveau, la correction des tests multiples (**cf. section 2.1.5**) est en effet nécessaire en IRMf. Malgré des procédures bien établies, Eklund *et al.* ont montré que certaines de ces procédures produisent des taux de faux positifs largement supérieurs au seuil de 5% habituellement choisi [61, 62].

Un frein à la réplicabilité des résultats est que le pipeline d'analyse est rarement détaillé en entier. Carp et Guo *et al.* ont ainsi analysé un grand nombre de publications d'IRMf et ont trouvé que certains détails de l'analyse sont rarement rapportés [38, 86]. Le détail du pipeline d'analyse est d'autant plus important qu'il existe une grande flexibilité d'analyse des données d'IRMf. Une chaîne peut en effet varier en fonction des prétraitements appliqués, des paramètres de ces prétraitements, de l'ordre des prétraitements et de la méthode d'analyse choisie. En analysant plus de 6 000 pipelines (prétraitement et analyse), Carp a trouvé que les résultats et notamment la localisation des pics d'activation varient en fonction du pipeline choisi [37]. Un autre problème est le report des tests d'hypothèses où il est courant de ne reporter que la p-valeur sans préciser ni la taille de l'effet, ni le nombre de degrés de liberté [43]. Or Chen *et al.* expliquent que la valeur statistique permet seulement une inférence binaire (hypothèse nulle contre hypothèse alternative) mais ne contient pas d'information sur l'amplitude de la réponse. Ils recommandent d'utiliser les valeurs statistiques pour ne faire apparaître que les régions significatives au-delà d'un seuil conservatif, et de colorer les cartes d'activation en fonction de la valeur de l'effet plutôt qu'avec la p-valeur.

1.3 Pourquoi l'IRMf des signaux alimentaires ?

Comprendre comment les stimuli alimentaires sont traités par le cerveau est un aspect essentiel pour comprendre la représentation des aliments ainsi que l'appréciation et le désir de nourriture qui contrôlent l'appétit et donc la prise alimentaire [26]. Or l'IRMf a permis d'identifier les régions cérébrales impliquées dans la perception des caractéristiques (saveur et intensité) des aliments et de leur valeur affective (goût plaisant ou déplaisant) [53, 196]. Cependant, la plupart des paradigmes visant à identifier les régions cérébrales impliquées dans le traitement des signaux alimentaires utilisent uniquement des images alimentaires. Ces paradigmes ont permis d'obtenir des réponses robustes dans plusieurs régions cérébrales comme l'insula, l'amygdale, le cortex orbito-frontal et le striatum (**cf. section A.2**) [213, 199, 214, 106]. Cependant, les caractéristiques temporelles de la réponse BOLD à des stimuli alimentaires, comme la forme ou la variabilité entre les répétitions de la réponse, ne sont complètement connues [153]. Une stimulation directe des sens chimiques par des arômes ou de la nourriture est obligatoire pour étudier la

chronologie de la dégustation qui impliquent à la fois des perceptions chimio-sensorielle et des processus cognitives. Une telle stimulation permet de plus l'étude des interactions olfacto-gustatives à l'origine de la perception multi-sensorielle de la saveur [193].

Mais la stimulation directe soulève plusieurs problèmes [226]. D'abord, les sujets doivent être allongés sur le dos en étant dans le scanner ce qui rend difficile la mastication. La mastication étant difficile, les aliments solides ne peuvent pas être utilisés et seuls les aliments liquides sont pris comme stimuli. De plus, comme la réponse cérébrale aux stimuli alimentaires survient quelques secondes après la prise des aliments, les aliments ne peuvent pas être pris avant d'entrer dans le scanner et doivent donc être administrés pendant que le sujet est dans le scanner. Ils sont en général délivrés par un gustomètre contrôlé soit par ordinateur soit manuellement [11]. Un gustomètre est composé de plusieurs seringues contenant les différents stimuli liquides, attachés à des tubes qui se terminent par un embout central, attaché à la tête de bobine. La plupart des études IRMf étudiant la réponse aux stimuli alimentaires utilisent un paradigme qui nécessite l'ingestion d'une petite quantité de solution contenant la saveur étudiée, même si certains gustomètres ont été conçus pour éviter la déglutition [83, 113]. Or la déglutition entraîne des mouvements important du corrélés avec la stimulation [25] et il est reconnu depuis longtemps que le risque de faux-positif peut augmenter lorsque le mouvement est corrélé au stimulus [87, 29, 129]. Les stimulations utilisant des liquides sont par les plus problématiques car le sujet doit successivement ouvrir et fermer la bouche, garder la boisson en bouche puis avaler ce qui génère des mouvements complexes des mâchoires, de la langue, du velum et du larynx [28]. Néanmoins, les stimuli gustatifs doivent être répétés au cours de l'expérience afin que leurs effets sur les activations atteignent une puissance statistique fiable [39]. En plus de ces mouvements corrélés à la stimulation, la déglutition crée une asynchronie car les récepteurs gustatifs sont stimulés dès que le liquide vient en contact de la bouche et de la langue, alors que les récepteurs olfactifs sont principalement activés par les éléments volatiles portés par voie rétro-nasale. Ce décalage entre le goût et l'odorat pose la question de savoir si les sensations du goût et de l'odeur sont perçues suffisamment proches dans le temps pour induire une intégration multi-sensorielle [166]. Comme les sujets avalent sur ordre visuel ou sonore et que la déglutition est une action rapide, le moment exact de la déglutition reste inconnu et seule une approximation peut être faite [99]. Deux méta-analyses récentes ont toutefois montré que les régions cérébrales impliquées dans la perception des caractéristiques des aliments (saveur et intensité) et de la valeur affective des aliments (goût plaisant ou déplaisant) ont été identifiées par plusieurs dizaines d'études IRMf [230, 231]. Cependant, ces deux méta-analyses ont mis en évidence une concordance modérée entre les études pour l'identification de ces régions cérébrales. Cette concordance n'excède pas les 60% de l'ensemble des expériences (le maximum étant atteint pour les régions codant l'intensité). On peut supposer que ce manque de concordance s'explique peut-être en partie par la présence de faux positifs dus aux mouvements [218].

Il est possible d'étudier par IRMf les régions cérébrales impliquées dans la représentation des qualités de l'aliment et des dimensions cognitives associées à cette prise alimentaire (plaisir, désir, choix,...) chez le consommateur. Cependant, les méta-analyses récentes ont montré que les résultats des études précédentes sont peu reproductibles [230, 231]. Il semble donc nécessaire de développer des méthodes d'analyse permettant à la fois de réduire le taux d'erreurs (faux positifs et faux négatifs) et de tenir compte des mouvements propres à la stimulation avec des aliments liquides. Le chapitre 2 décrit les principales méthodes utilisées pour l'analyse de données d'IRMf.

Chapitre 2

Analyse des données d'IRM fonctionnelle

Il existe plusieurs méthodes d'analyse des données d'IRMf, qui se séparent en deux catégories les méthodes fondées sur un modèle et les méthodes conduites par les données. Le modèle linéaire général appartient à la première catégorie : en effet, il nécessite de connaître le paradigme afin d'identifier des régresseurs d'intérêts qui correspondent aux phases de l'expérience à modéliser. C'est à partir de ces régresseurs que sont construits les tests statistiques. À l'inverse, les méthodes d'analyse conduites par les données ne nécessitent pas de modélisation *a priori*. Elles travaillent à partir des données et en extraient des composantes. Avant l'analyse, les données d'IRMf sont soumises à une chaîne de prétraitements qui ont été brièvement décrit en section 1.1.3.

2.1 Analyse fondée sur les modèles

Les méthodes classiques de détection des activations se fondent sur la définition d'un modèle *a priori*. Des hypothèses construites à partir du modèle sont ensuite testées pour déterminer quels voxels sont activés par la stimulation. On distingue deux types d'analyse par des modèles : les modèles massivement univariés où le modèle est estimé et l'hypothèse est testée pour chaque voxel et les modèles multivariés dont le but est de déterminer quelle information est encodée par une région cérébrale.

2.1.1 Le modèle linéaire général (GLM)

Le modèle linéaire général (GLM) est un modèle statistique linéaire qui généralise la régression linéaire à plusieurs régresseurs/prédicteurs et à plusieurs variables observées. Le GLM est un modèle car il formalise mathématiquement la connaissance d'une relation entre les données observées $\boldsymbol{X}$ et des données connues $\boldsymbol{Y}$ [168]. Il est de plus linéaire c'est-à-dire que les données $\boldsymbol{X}$ sont formées d'une somme pondérées des données connues $\boldsymbol{Y}$ appelées régresseurs [168]. Il est général car les régresseurs peuvent être des données mesurées mais également des appartenances à une condition ou un groupe [168]. Il s'écrit sous forme mathématique $\boldsymbol{X} = \boldsymbol{Y}.\beta + \epsilon$ où $\boldsymbol{X}$ est une matrice contenant les données observées une colonne par donnée, $\boldsymbol{Y}$ est la matrice *design* décrivant le modèle utilisé, β représentent les paramètres estimés et ϵ est le résidu c'est-à-dire la différence entre les données observées et le modèle estimé par $\boldsymbol{Y}.\beta$ qui suit une loi définie par le modèle. Sous forme matricielle, le modèle linéaire général se réécrit donc :

$$\begin{pmatrix} \boldsymbol{x}_1 & \cdots \boldsymbol{x}_j & \cdots \boldsymbol{x}_{n_\mathrm{V}} \end{pmatrix} = \begin{pmatrix} Y_{11} & \cdots & Y_{1n_\mathrm{R}} \\ \vdots & \ddots & \vdots \\ Y_{n_\mathrm{T}1} & \cdots & Y_{n_\mathrm{T}n_\mathrm{R}} \end{pmatrix} \begin{pmatrix} \beta_{11} & \cdots & \beta_{1n_\mathrm{V}} \\ \vdots & \ddots & \vdots \\ \beta_{n_\mathrm{R}1} & \dots & \beta n_\mathrm{R} n_\mathrm{V} \end{pmatrix} + \begin{pmatrix} \epsilon_1 & \cdots \epsilon_j & \cdots \epsilon_{n_\mathrm{V}} \end{pmatrix}$$

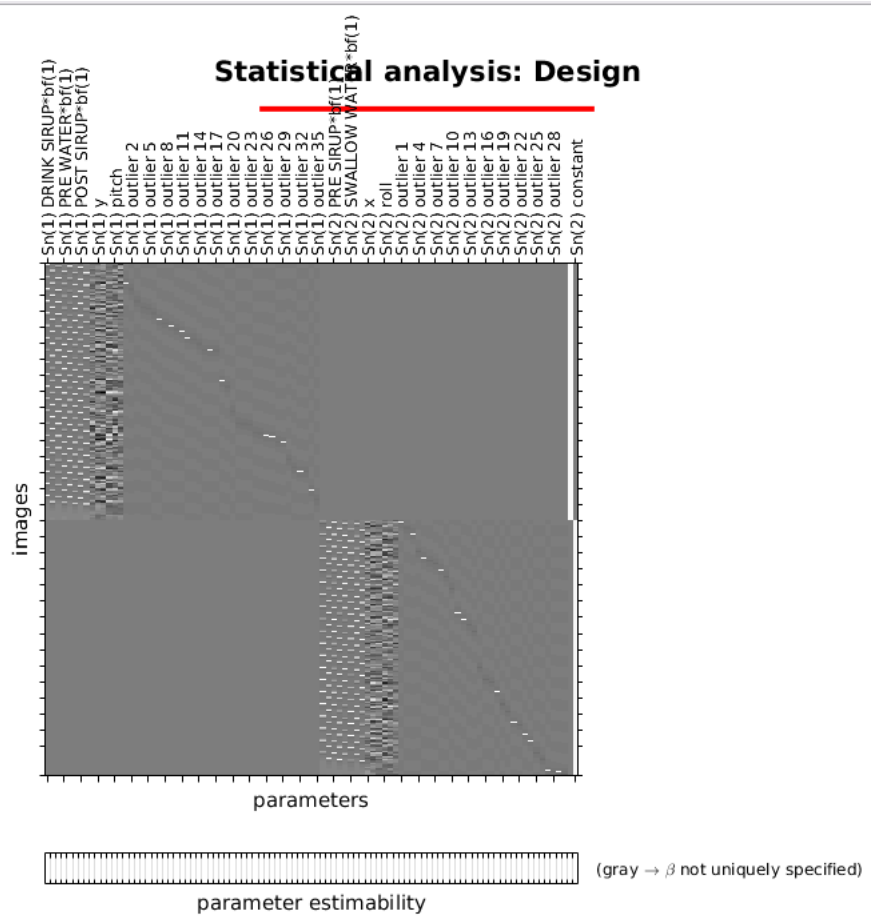


FIGURE 2.1 – Exemple de matrice *design* pour un paradigme gustatif .

avec n_{V} le nombre de voxels, n_{T} le nombre de points temporels et n_{R} le nombre de régresseurs. En IRMf, chaque colonne $\boldsymbol{x}_j$ de la matrice $\boldsymbol{X}$ représente le signal BOLD temporel d'un voxel. La matrice *design* $\boldsymbol{Y}$, quant à elle, regroupe plusieurs composants qui expliquent les données observées (**cf. Figure 2.1**). Les paramètres β définissent la contribution de chaque composante de la matrice *design* à la valeur de $\boldsymbol{X}$. Ils sont estimés pour minimiser l'erreur ϵ par la méthode des moindres carrés $\beta = (\boldsymbol{Y}^\top \boldsymbol{Y})^{-1} \boldsymbol{Y}^\top \boldsymbol{X}$.

Le modèle linéaire général sert pour l'analyse individuelle une fois que le paradigme de l'expérience a été modélisé. La modélisation du paradigme permet de définir les régresseurs d'intérêt du modèle. À ces régresseurs, peuvent être ajoutés des régresseurs de non intérêt. En IRMf, les paramètres de recalage rigide sont très souvent utilisés comme régresseurs de non intérêt. Le logiciel SPM (Statistical Parametric Mapping, Wellcome Department of Cognitive Neurology, Londres, Royaume-Uni) ajoute automatiquement la moyenne du signal comme régresseur constant. Ces régresseurs sont convolués avec une ou plusieurs fonctions modélisant la réponse hémodynamique (HRF). Une régression linéaire est ensuite effectuée pour estimer les β pour chaque régresseur. En IRMf, les β sont sous forme des cartes cérébrales 3D.

2.1.2 Détection d'une activation univariée

L'idée sous-jacente du test univarié est qu'un ensemble de voxels (regroupant un très grand nombre de neurones puisque la résolution est grossièrement millimétrique) partagent la même taille de l'effet. Le test mis en œuvre a pour objet de tester si l'effet est différent de zéro, statistiquement parlant. Tester s'il y a un effet d'un facteur dans le modèle revient soit à tester si une combinaison de bêtas est nulle ($\lambda^\top \beta = 0$)) soit à tester si une combinaison de bêtas est

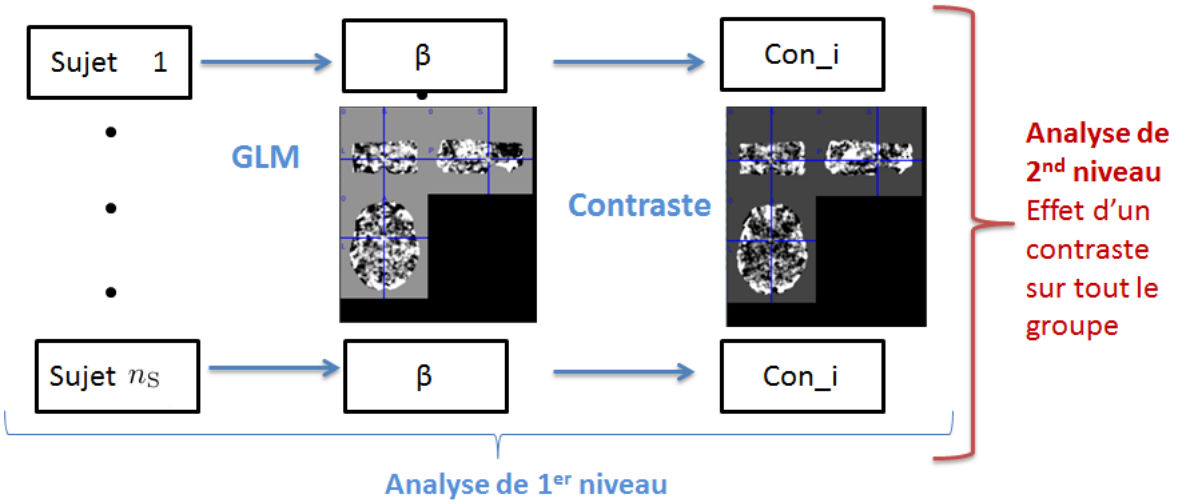


FIGURE 2.2 – Analyses de premier et second niveau avec des données d'IRMf.

strictement positive/négative ($\lambda^\top \beta > 0$) ou $\lambda^\top \beta < 0$)). Si l'hypothèse nulle ne concerne qu'une seule combinaison linéaire de bêtas, le rejet ou l'acceptation de hypothèse nulle s'effectue alors à l'aide d'un test-T ou d'un test-F calculé comme suit [168] :

$$t_{df} = \frac{\lambda^\top \beta}{\sigma(\lambda^\top \beta)} \ ,$$

où $\sigma(\lambda^\top \beta)$ est l'écart-type de $\lambda^\top \beta$. Si les erreur sont indépendantes et suivent une loi gaussienne, le test t_{df} suit alors une loi de Student. Pour tester conjointement plusieurs colonnes de la matrice $\boldsymbol{Y}$, le test-F se forme en comparant la somme au carré des résidus calculés avec le modèle réduit (autrement dit sans les régresseurs correspondant aux effets testés) et la somme au carré des résidus calculés avec le modèle complet. Le test est effectué pour l'ensemble des données observées (donc pour chaque voxel en IRMf).

2.1.3 Détection d'une activation multivariée

2.1.3.1 Principes

À l'inverse du GLM où chaque voxel est une variable indépendante (des autres voxels), les méthodes multivariées ne considèrent plus les voxels comme des variables indépendantes mais tiennent compte de la relation entre les voxels. Les analyses multivariées (*Multivariate pattern analysis* (MVPA)) sont utilisées dans deux buts différents, soit pour prédire une condition à partir du signal de plusieurs voxels, soit pour détecter les régions cérébrales codant le contraste entre les conditions [95]. Ces deux utilisations ont donc des objectifs très différents. La prédiction vise à établir des biomarqueurs pouvant être utilisés pour prédire un état particulier du cerveau. La détection correspond à l'objectif visé dans ce travail et consiste à interpréter la manière dont le cerveau code l'information temporellement et géographiquement.

L'exemple donné par Haynes [94] illustre ce second cas. Il s'agit de déterminer les territoires du cerveau impliqués dans le codage de l'information contenue dans deux images différentes. Les figures 2.3.A et 2.3.B montrent le signal acquis dans plusieurs voxels pendant le visionnage d'une image de chat et le visionnage d'une image de chien. Chaque dimension représente un voxel (il y a donc autant d'axes que de voxels considérés). Le classifieur a pour but de séparer les deux distributions (l'une pour le visionnage d'une image de chat et l'autre pour le visionnage d'une image de chien pour notre exemple) en évitant de sur-apprendre. Pour cela, il détermine un hyper-plan qui sert de séparateur entre les points en groupe distincts. La figure 2.3.C illustre le

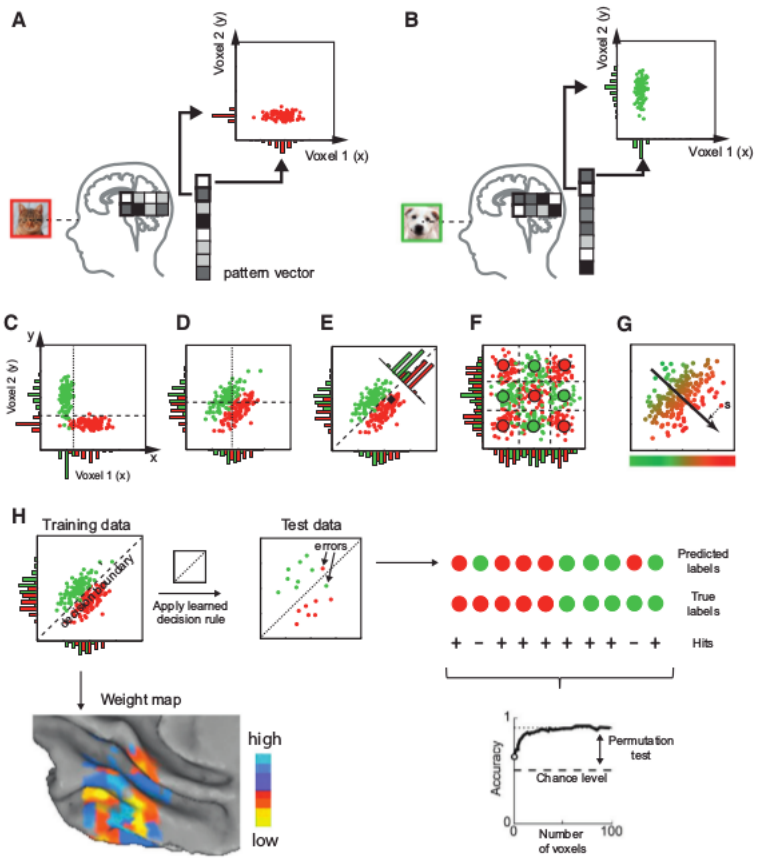


FIGURE 2.3 – Analyse de motifs multivarié issue de [94].

cas où les deux distributions sont séparables en utilisant un seul axe (séparateurs en pointillé parallèles aux axes) car les distributions ne sont pas superposées. Le cas général est montré en figure 2.3.D : les deux distributions sont superposées. Il faut alors prendre en compte les valeurs acquises dans tous les voxels de manière simultanée. Une première approche est de déterminer une frontière de décision linéaire à l'aide d'une analyse de discriminant linaire (*Linear discriminant analysis*) ou d'une machine à vecteurs de support (SVM) linéaire. La figure 2.3.E montre un tel séparateur en pointillé. Dans certains cas illustrés en figure 2.3.F, les distributions ne peuvent pas être séparées par une frontière linéaire et des approches non linéaires comme les SVMs non linéaires ou la méthode de plus proches voisins (*k-nearest neighbours*). La figure 2.3.G illustre la prédiction d'une variable continue (plutôt que discrète).

Le classifieur fonctionne en deux étapes (**cf. Figure 2.3.H**) l'apprentissage et le test. Pendant l'apprentissage, seule une partie des données est utilisée. Cette phase consiste à trouver une frontière de décision qui sépare les données d'apprentissage en groupe distinct. Cette frontière est ensuite testée sur les données de tests (les données qui n'ont pas été utilisées pour l'apprentissage) : la proportion d'images bien classées donne un score d'exactitude. Dans notre exemple, ce serait le pourcentage de volumes acquis pendant le visionnage d'une image de chat à qui le classifieur a attribué le label « chat ». Ces deux étapes sont généralement répétées plusieurs fois en partitionnant différemment les données entre l'étape d'apprentissage et l'étape de test. Cette répétition des deux phases est appelée validation croisée. L'exactitude est alors moyennée sur les différentes itérations. La validation croisée à k-couches consiste à découper les données en k sous-ensembles puis d'entraîner le classifieur sur $k-1$ ensemble et de tester le classifieur sur le sous-ensemble restant (sur lequel le classifieur n'a pas été entraîné). Grootswagers *et al.* recommandent également la validation croisée en laissant dehors un exemplaire : toutes les répétitions

correspondant à une condition (le visionnage d'image de chat par exemple) servent dans la phase de test et les répétitions correspondant aux autres conditions (visionnages d'image de chien) sont utilisées pour entraîner le classifieur, et le mécanisme est répété pour chaque condition [85]. Dans leur cas (animé versus inanimé), cette validation croisée séparant les conditions leur permet d'être sûr que le classifieur se fonde sur des critères traduisant que l'objet est animé, et non sur des critères visuels. Les données d'entraînement et les données de test doivent être indépendantes afin d'éviter le sur-ajustement du prédicteur et les inférences circulaires.

Si le classifieur prédit mieux que la chance (dans notre exemple, il y a deux labels donc une chance sur deux d'attribuer le bon label), cela prouve que le classifieur peut généraliser ce qu'il a appris à de nouvelles données. Pour cela, il faut ensuite déterminer si ces scores de prédiction sont statistiquement différents de la chance. Hebart *et al.* soulignent que la distribution des scores de prédiction ne suivent pas une loi binomiale et donc les tests binomiaux ne sont pas adaptés. Une alternative est alors d'utiliser des permutations c'est-à-dire de mélanger les données et de calculer à nouveau les cartes de précision sur les données mélangées. L'interprétation des cartes de précision est difficile car elles ne permettent pas de conclure qu'un voxel donné contribue de manière significative au décodage. En effet, le poids d'un voxel reflète l'utilité du voxel en interaction avec les autres voxels pour la classification [95]. Pour tester si un voxel contribue de manière significative, il faut tester s'il y a une différence significative en incluant le voxel et en l'excluant [94]. De plus, un voxel peut ne pas contenir d'information mais être utile à la classification parce qu'il permet de débruiter le signal par exemple. Comme le soulignent Hebart *et al.*, l'analyse multivariée ne dépend pas seulement de ce qui est traité classiquement comme du signal mais également de ce qui est considéré comme bruit. De plus, l'information mesurée par un classifieur peut aussi bien refléter des différences dans les moyennes mutlivariées que des différences de variabilité [95].

2.1.3.2 Que mettre en entrée dans le classifieur ?

Dans le cas de données d'IRMf, il faut choisir quelle partie des données sont passées dans classifieur. Une approche naturelle est de considérer l'ensemble des voxels associés à la série temporelle entière. Comme le souligne Hebart *et al.*, l'avantage est que le classifieur dispose de toute l'information disponible, cependant il est par la suite difficile de dire l'origine de l'information [96]. De plus, comme il y a beaucoup plus de voxels que de volumes acquis, le classifieur peut souffrir du fléau de la dimension c'est-à-dire que le le pouvoir prédictif du classifieur est réduit. C'est pourquoi Allefeld *et al.* suggèrent d'employer en entrée du classifieur les bêtas issus d'un modèle linéaire général [6]. Cela permet de réduire l'information temporelle et de prendre en compte la situation où des stimulations sont effectuées avant de revenir à l'état de base (c'est-à-dire lorsque le temps entre deux stimuli successifs est inférieur à la durée de la HRF). En outre, les cartes de bêtas ont un label clair, alors qu'un volume EPI peut correspondre à un mélange de différentes conditions expérimentales.

Pour ne pas considérer tous les voxels, Hebart *et al.* proposent plusieurs approches [96]. La plus utilisée est la *searchlight analysis*, qui peut se voir comme un ensemble de régions d'intérêt sphériques, mais sans sélectionner les régions d'intérêt ce qui évite le biais par rapport à la localisation de l'information. Toutefois, cette analyse ne tient pas compte non plus de l'information codée entre des régions cérébrales distantes. Concrètement, le classifieur est appliqué à un petit ensemble de voxels, par exemple une sphère d'un rayon donné autour d'un voxel choisi, et l'analyse est répétée en prenant différents voxels comme centre de la sphère. La deuxième approche proposée est de sélectionner des régions d'intérêt. Toutefois, les régions d'intérêt de tailles variées sont difficiles à comparer et l'information encodée entre les différentes régions d'intérêt n'est pas prise en compte par le classifieur.

Loula *et al.* proposent un décodage dans le domaine temporel pour des expériences de type *event related* où il est possible d'enchaîner vite les stimulations.[135]. Plutôt que de classifier les bêtas issus du GLM, ils proposent de séparer les données en données d'apprentissage et données

de test puis d'effectuer un GLM sur les données d'apprentissage et de calculer la matrice design sur les données de test à partir de l'estimation du GLM puis la classe de chaque stimulation est prédite.

2.1.4 Inférences de groupe

Les inférences de groupe consistent à chercher des voxels qui présentent des différences significatives entre régresseurs de manière consistante sur le groupe afin d'inférer à l'échelle de la population. On distingue deux types d'analyses pour inférer à l'échelle du groupe : l'analyse à effet fixe et l'analyse par effets aléatoires. L'analyse à effet fixe considère le facteur sujet comme les autres régresseurs du modèle et ne tient compte que de la variance intra-sujet due à la répétition des stimuli comme source de variance. *A contrario*, l'analyse par effets aléatoires considère les sujets comme un facteur aléatoire, tirés au hasard dans la population. En plus de la variance intra-sujet, elle tient également compte de la variance inter-sujet. L'analyse par effet fixe ne permet pas de généraliser les résultats de l'inférence de groupe à l'ensemble de la population [154]. Pour l'analyse multivariée, Hebart *et al.* suggèrent de combiner les cartes de prédiction individuelle et de mener une analyse de second niveau comme pour le GLM classique et d'utiliser alors un test-T [96].

2.1.5 Correction des tests multiples par le FDR

Les tests statistiques présentés en section 2.1.2 et 2.1.3.1 sont effectuées sur l'ensemble des voxels donc les tests sont répétés entre 10^4 et 5.10^5fois. La détection des zones activées se fait à travers un seuillage du score statistique calculé pour chaque voxel. Le plus souvent en IRMf, plutôt que de donner la valeur du test statistique, les articles rapportent la valeur-P [43] qui est la probabilité d'obtenir sous l'hypothèse nulle H_0 un score égal ou plus extrême que le score observé sur l'échantillon [223, 41]. Pour un seuil fixé α, si la valeur-P est inférieure à ce seuil, alors on rejette l'hypothèse nulle (pas de variation significative du signal dans le voxel) ; sinon on accepte l'hypothèse nulle [156]. Le choix du seuil α permet de contrôler les faux positifs. Un faux positif est lorsqu'on accepte l'hypothèse nulle alors qu'elle est fausse.

Cependant l'utilisation du seuil $\alpha = 0,05$ sans correction peut fausser les analyses : ainsi, une étude a trouvé une activité cérébrale significative dans le cerveau d'un saumon mort [23]. Un moyen de corriger le seuil est d'utiliser la correction de Bonferroni : pour un seuil α, si on effectue n tests indépendants, le seuillage des valeur-P se fait avec le seuil α/n. Cette correction est délaissée maintenant car elle se révèle trop dépendante du nombre de voxels, conservative sur le cerveau entier et libérale sur des régions ciblées. La correction pour les tests multiples s'effectue soit en contrôlant le *Familywise Error Rate* (FWER) qui est la probabilité qu'il y ait un faux positif. [155] soit le *False Discovery Rate* (FDR) qui est la proportion de faux positifs parmi les tests positifs [140, 22]. Contrôler le FWER à 0,05 signifie qu'il y a 5% de chance qu'il y ait un ou plusieurs faux positifs parmi tous les tests. Contrôler le FDR à 0.05 signifie qu'il y a 5 % de faux positifs parmi les résultats observés [24]. Le contrôle du FDR permet une analyse plus sensible des données que les procédures conventionnelles de contrôle du FWER. En effet, si toutes les hypothèses nulles sont vraies alors le FDR est équivalent au FWER et sinon, le FDR est inférieur ou égal au FWER. Par conséquent, toute procédure qui contrôle le FWER contrôlera également le FDR. En revanche, l'inverse est faux et donc une procédure qui contrôle le FDR peut être plus puissante qu'une procédure contrôlant le FWER [22]. Actuellement, la démarche de référence [155, 22, 24, 97, 44, 45] est donc de contrôler le FDR.

2.1.6 Critiques des méthodes fondées sur des modèles

Le GLM est la méthode d'analyse la plus utilisée en IRMf. En effet, il présente une grande flexibilité qui permet d'adopter plusieurs stratégies de test [147]. Cependant, les modèles créés

sont rarement validés bien qu'il existe plusieurs initiatives pour valider un modèle. Luo *et al.* ont proposé une boîte à outil dans SPM pour vérifier la validité des hypothèses du modèle comme la normalité ou l'homoscédasticité [137], mais qui n'est désormais plus maintenue [189]. Plus récemment, Soch *et al.* ont introduit une méthode pour comparer plusieurs modèles [190] qu'ils ont ensuite intégré dans une boîte à outil pour SPM permettant la validation d'un modèle, la comparaison de plusieurs modèles et la sélection d'un modèle parmi plusieurs [189].

Le non respect des hypothèses du modèle linéaire peut venir de plusieurs facteurs [147]. Le premier facteur est la non-linéarité du signal BOLD due à la réponse vasculaire [31] et au comportement adaptatif de la réponse neuronale [133]. Cependant, la non-linéarité peut ne pas être prise en compte lorsque les stimuli sont séparés par un intervalle d'au moins quatre secondes. De plus, la présentation aléatoire des stimuli, l'ajout d'un jitter aléatoire à la durée de stimulation et l'augmentation du nombre de répétition atténuent l'effet non linéaire [147]. Le deuxième facteur est l'hétéroscédasticité peu étudiée en IRMf excepté par Luo *et al.* qui ont trouvé une violation de l'hypothèse d'Homoscédasticité du GLM, due aux artéfacts [137]. Le troisième facteur est dû à la mauvaise modélisation de la réponse hémodynamique. Il a en effet été montré que la HRF est variable entre les sujets, et pour un même sujet la HRF varie d'une tâche à l'autre, d'une région à l'autre, voire d'un jour à l'autre [5, 89], rendant la modélisation difficile. Le quatrième facteur est la mauvaise spécification du modèle avec les risques (potentiellement cumulés) de la sur-modélisation consistant à inclure un régresseur non nécessaire qui va alors ajuster le bruit, et de la sous-modélisation consistant à oublier d'ajouter un régresseur nécessaire dans le modèle [190]. Mais, il est difficile de prendre en compte des facteurs individuels relatifs à l'état physiologique du sujet et la manière dont il effectue les taches. Or on sait que le niveau d'activité dépend par exemple de l'attention [217], de l'habituation, de l'apprentissage [56], du sommeil [73] ou de l'ingestion de caféine [130]. De plus, même en alternant de manière aléatoire les stimulations, les réponses cérébrales consécutives à une même stimulation sont variables dans le temps car sujettes à des mécanismes d'adaptation complexes [150, 112]. Or les régresseurs ne doivent pas être colinéaires entre eux c'est-à-dire qu'aucun des régresseurs n'est corrélé de manière parfaite avec un autre régresseur ou une combinaison d'autres régresseurs. Mumford *et al.* ont en effet montré que l'orthogonalisation des régresseurs pour s'affranchir de la colinéarité n'est pas forcément la meilleure solution et qu'il est préférable de construire un modèle sans colinéarité [152].

Le cinquième facteur vient de l'auto-corrélation présente dans le signal mesuré. Cette auto-corrélation peut toutefois être corrigée par des algorithmes de pré-blanchiment [147]. Cependant, les différents algorithmes peuvent aboutir à des inférences différentes [121]. Le sixième facteur vient du fait que le signal BOLD ne suit pas une loi gaussienne : Hanson *et al.* ont en effet montré que le signal BOLD suivrait plutôt une loi Gamma [90]. Un dernier facteur est la mauvaise prise en compte du bruit. Or le signal observé dans chaque voxel est constitué en plus du signal d'intérêt de plusieurs autres sources de signal comme les pulsations cardiaques et respiratoires, le mouvement ou la dérive temporelle du scanner. Ces différentes sources de bruit rendent difficile l'identification du signal BOLD d'intérêt [145].

Le problème de la mauvaise spécification du modèle est qu'elle peut conduire à une perte de la puissance statistique et une augmentation du taux de faux positif supérieur au niveau nominal [134, 128]. Récemment, Gonzales-Castillo *et al.* ont en effet montré que les erreurs de modélisation dominent les autres sources de variance intra-sujet dans l'ensemble du cerveau et correspondent entre 60% et 80% de la variance intra-sujet [80]. La part de variance expliquée est donc minoritaire ce qui signifie que la modélisation est inadéquate. Pour réduire la variance intra-sujet, Gonzales-Castillo *et al.* préconisent de mieux prendre en compte la variabilité spatiale de la réponse hémodynamique et de modéliser des composantes liées aux tâches supplémentaires comme les réponses transitoires [80].

2.2 Méthodes conduites par les données

Les articles les plus récents d'analyse de données issues de l'IRMf se tournent de plus en plus vers des méthodes dites conduites par les données qui présentent l'avantage de ne pas nécessiter de modèle et peuvent donc prendre en compte plus de variance, en étant moins tributaire de modèles fixés a priori.

2.2.1 Analyse en composantes principales (ACP)

L'analyse en composantes principales est une procédure qui convertit un ensemble d'observations qui peuvent être corrélées entre elles en un nouvel ensemble de variables linéairement non corrélées appelées composantes principales, au moyen d'une transformation orthogonale. Le nombre de composantes principales est plus petit ou égal au nombre de variables initiales. Les composantes principales forment une base de vecteurs orthogonaux non corrélées. Considérons une matrice $\boldsymbol{X}$ de taille $n_{\mathrm{T}} \times n_{\mathrm{V}}$ où les lignes correspondent aux observations (qui en IRMf correspondent au signal image ou aux paramètres issus de la modélisation) et les colonnes à des caractéristiques. Considérons de plus que chaque colonne de $\boldsymbol{X}$ a une moyenne nulle. Mathématiquement, on définit l'analyse en composantes principales par un ensemble de vecteurs de dimension p de poids $\boldsymbol{w}_k$ qui fait correspondre chaque ligne $\boldsymbol{x}_{(i)}$ de $\boldsymbol{X}$ à un nouveau vecteur de composantes principales $\boldsymbol{t}_i$ donné par $\boldsymbol{t}_{k(i)} = \boldsymbol{x}_{(i)}.\boldsymbol{w}_{(k)}$. La première composante est définie par

$$\boldsymbol{w}_{(1)} = \arg\max \left\{ \frac{\boldsymbol{w}^\top \boldsymbol{X}^\top \boldsymbol{X} \boldsymbol{w}}{\boldsymbol{w}^\top \boldsymbol{w}} \right\} .$$

La k^{ème} composante principale est ensuite calculée en soustrayant les k-1 composantes principales de $\boldsymbol{X}$

$$\hat{\boldsymbol{X}}_k = \boldsymbol{X} - \sum_{s=1}^{k-1} \boldsymbol{X} \boldsymbol{w}_{(s)} \boldsymbol{w}_{(s)}^\top$$

puis en trouvant le vecteur qui maximise la variance restante

$$\boldsymbol{w}_{(k)} = \arg\max \left\{ \frac{\boldsymbol{w}^\top \hat{\boldsymbol{X}}_k^\top \hat{\boldsymbol{X}}_k \boldsymbol{w}}{\boldsymbol{w}^\top \boldsymbol{w}} \right\} .$$

Les composantes principales peuvent être obtenues par la décomposition en valeurs propres de la matrice $\boldsymbol{X}\boldsymbol{X}^\top$ et correspondent aux vecteurs propres et la proportion de variance expliquée par la composante principale est donnée par le ratio de la valeur propre associée au vecteur propre sur la somme des valeurs propres. L'analyse en composantes principales peut être utilisées pour réduire les données. En effet, on peut ne pas garder toutes les composantes principales. La réduction des données consiste à ne garder que les composantes principales qui expliquent un certain pourcentage de variance. L'analyse en composantes principales (ACP) a été utilisée en IRMf pour étudier la connectivité fonctionnelle [122], pour explorer les données [216, 17], pour débruiter les données [21, 191] et principalement pour réduire la dimension des données [188]

2.2.2 Analyse en composantes indépendantes (ACI)

2.2.2.1 Principe de l'analyse en composantes indépendantes

L'analyse en composantes indépendantes est une technique de séparation des sources à l'aveugle [100, 1] car il s'agit de trouver n_{R} sources à partir des données observées. Notons $\boldsymbol{x}_i(t)\, i \in \{1, \ldots, n_{\mathrm{V}}\}, t \in \{1, \ldots, n_{\mathrm{T}}\}$ les variables observées. L'analyse en composantes indépendantes (ACI) suppose que ces variables observées sont la combinaison linéaire de variables

cachées $\boldsymbol{s}_j(t)\, j \in \{1, \ldots, n_{\mathrm{R}}\}$ avec des coefficients inconnus a_{ij}. On peut donc écrire

$$\forall i \in \{1, \ldots, n_{\mathrm{V}}\}, \boldsymbol{x}_i(t) = \sum_{j=1}^{n_{\mathrm{R}}} a_{ij} \boldsymbol{s}_j(t) \ .$$

L'ACI détermine donc ,à partir des seules observations connues, à la fois les $\boldsymbol{s}_j(t)$ qui sont les composantes indépendantes et les a_{ij} appelés coefficients de mélange. Si on range les observations $\boldsymbol{x}_i$ dans une matrice $\boldsymbol{X}$ de taille $n_{\mathrm{V}} \times n_{\mathrm{T}}$, les composantes indépendantes dans une matrice $\boldsymbol{S}$ de taille $n_{\mathrm{R}} \times n_{\mathrm{T}}$ et les coefficients de mélange dans une matrice $\boldsymbol{A}$ de taille $n_{\mathrm{V}} \times n_{\mathrm{T}}$, on peut écrire l'équation de l'ACI : $\boldsymbol{X} = \boldsymbol{AS}$. Les matrices $\boldsymbol{A}$ et $\boldsymbol{S}$ peuvent être identifiées à deux ambiguïtés près lorsqu'on pose l'hypothèse que les composantes indépendantes sont non gaussiennes. Plus précisément, on suppose :

1. Les composantes s_j sont mutuellement indépendantes statistiquement (l'indépendance sert de critère pour distinguer les sources). Autrement dit, leur densité de probabilité jointe est factorisable en fonction des densités de probabilité marginale $p_j(s_j)$:

$$p(\boldsymbol{s}_1, \ldots, \boldsymbol{s}_{n_{\mathrm{R}}}) = \prod_j p_j(\boldsymbol{s}_j) \ .$$

2. Les composantes s_j ont des distributions non-gaussiennes (non-normales).
3. La matrice de mélange A est carrée et inversible

Dans ce cas, on peut déterminer $\boldsymbol{A}$ et $\boldsymbol{S}$ à deux ambiguïtés près. En effet, on ne peut pas déterminer la variance des composantes indépendantes car la multiplication d'une source $\boldsymbol{s}_j$ par un scalaire peut être compensée par la division de la $j^{ème}$ colonne $\boldsymbol{A}_j$ de $\boldsymbol{A}$ puisque $\boldsymbol{A}$ et $\boldsymbol{S}$ sont toutes les deux inconnues. De plus, l'ordre des composantes est indéterminée : on peut librement changer l'ordre des termes dans la combinaison linéaire et donc permuter l'ordre des sources dans les matrices $\boldsymbol{A}$ et $\boldsymbol{S}$. La troisième hypothèse n'est pas nécessaire et peut être simplifiée de différentes manières.

Concrètement, les algorithmes ACI divisent l'estimation du modèle en deux étapes : un blanchiment préliminaire des données et l'estimation des composantes indépendantes [101]. Le blanchiment consiste à transformer les données $\boldsymbol{X}$ par une matrice $\boldsymbol{V}$ telle que $\boldsymbol{Z} = \boldsymbol{VX}$ soit blanche, ce qui est défini par

$$\frac{1}{T}\boldsymbol{Z}\boldsymbol{Z}^{\top} = \boldsymbol{I}$$

où $\boldsymbol{I}$ est la matrice identité. Cette première étape peut se faire à l'aide d'une analyse en composantes principales. La deuxième étape consiste ensuite à estimer $\boldsymbol{A}$ en maximisant une fonction objectif qui mesure la non-gaussianité. Une première mesure de la non-normalité des données est la négentropie définie par J($\boldsymbol{x}$)=H($\boldsymbol{x}_{gauss}$)-H($\boldsymbol{x}$) où $\boldsymbol{x}$ est une variable aléatoire ; $\boldsymbol{x}_{gauss}$ est une variable aléatoire gaussienne dont la matrice de covariance est égale à la matrice de covariance de $\boldsymbol{x}$, et H($\boldsymbol{x}$) est l'entropie de $\boldsymbol{x}$. Cette entropie se définit pour une variable aléatoire $\boldsymbol{x}$ de densité f($\boldsymbol{x}$) par

$$H(\boldsymbol{x}) = \int f(\boldsymbol{x}) \log f(\boldsymbol{x}) d\boldsymbol{x} \ .$$

La négentropie est toujours positive et est nulle si et seulement si $\boldsymbol{x}$ a une distribution normale. L'algorithme fastICA implémente l'analyse en composantes indépendantes en maximisant une approximation de la négentropie. On peut également maximiser la non-normalité en minimisant l'information mutuelle. On définit cette information mutuelle entre n_{R} variables aléatoires (x_i) scalaire par

$$I(x_1, \ldots, x_{n_{\mathrm{R}}}) = \sum_{i=1}^{n_{\mathrm{R}}} H(x_i) - H(\boldsymbol{x}) \ .$$

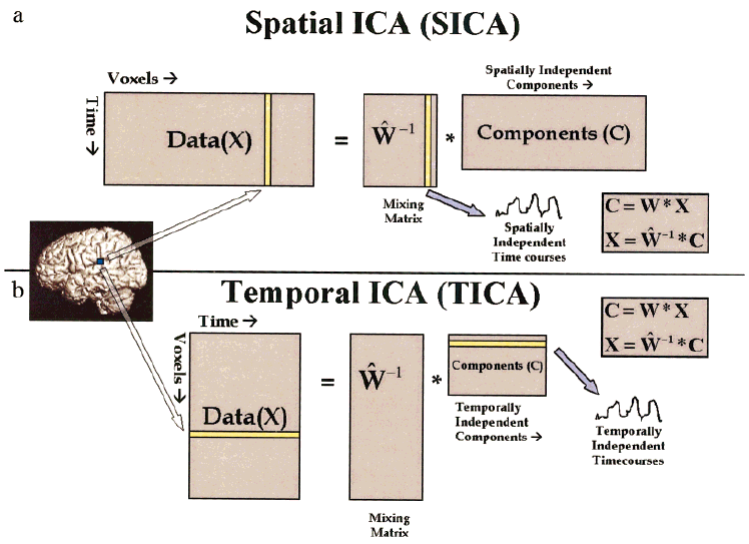


FIGURE 2.4 – Illustration de l'ACI spatiale et de l'ACI temporelle issue de [35].

On peut montrer que

$$I(x_1, \ldots, x_{n_R}) = c - \sum_i J(x_i)$$

où c est une constante. L'algorithme Infomax utilise l'information mutuelle comme fonction objectif [120].

2.2.2.2 Application aux données d'IRMf

On distingue deux types d'ACI applicables aux données d'imagerie fonctionnelle : une analyse en composantes indépendantes spatiales et une analyse en composantes indépendantes temporelles [35]. Dans l'ACI spatiale, on suppose que les données observées sont représentées par une matrice $\boldsymbol{X}$ de taille $n_T \times n_V$ où n_T est le nombre de points temporels et n_V le nombre de voxels, les signaux sont les n_V voxels spatiaux aplatis dans un vecteur 1D et il y a donc n_T observations différentes de ces signaux : une observation à chaque pas de temps. La décomposition de l'ACI spatiale (**cf. figure 2.4**) s'écrit $\boldsymbol{S} = \boldsymbol{WX}$ où $\boldsymbol{W}$ est l'inverse de la matrice $\boldsymbol{A}$ de taille $n_R \times n_R$, appelée matrice de déminage, contenant les n_R composantes indépendantes ; ou encore $\boldsymbol{X} = \boldsymbol{W}^{-1}\boldsymbol{S}$ où les composantes spatialement indépendantes sont situées dans les lignes de $\boldsymbol{S}$ et les courbes temporelles spatialement indépendantes associées sont dans les colonnes de W^{-1}. À l'inverse, dans l'ACI temporelle, les données observées sont de taille $n_V \times n_T$ et la décomposition s'écrit $\boldsymbol{S} = \boldsymbol{WX}$ où $\boldsymbol{W}$ est la matrice de déminage de taille $n_R \times n_R$ et $\boldsymbol{S}$ une matrice n_R par n_T contenant les n_R composantes indépendantes. Cette décomposition se réécrit comme précédemment $\boldsymbol{X} = \boldsymbol{W}^{-1}\boldsymbol{S}$ où les courbes temporelles temporellement indépendantes se trouvent dans les lignes de $\boldsymbol{S}$ et les cartes associées (temporellement indépendantes) sont dans les colonnes de $\boldsymbol{W}^{-1}$. En IRMf, les régions se distinguent selon la réponse neuronale au paradigme. De ce fait, il parait plus naturel de séparer les sources selon un critère d'indépendance temporelle. Pour autant, l'ACI spatiale prévaut car l'ACI est plus efficace algorithmiquement. Lorsque leurs hypothèses sont respectées, ces deux méthodes donnent des résultats similaires. Cependant, l'ACI spatiale ne peut être appliquée que lorsque les sources sont spatialement indépendantes, alors que l'ACI temporelle suppose des sources temporellement indépendantes. Néanmoins, Gao *et al.* ont montré que l'ACI temporelle et spatiale ne sont pas vraiment équivalentes et suggèrent que l'ACI temporelle est potentiellement plus intéressante que l'ACI classiquement utilisée. Le nombre de composantes indépendantes est souvent déterminé à l'aide soit du critère d'information d'Akaike

(AIC) soit du critère de longueur minimale de description (MDL) [33] :

$$AIC(n_{\mathrm{R}}) = -2n_{\mathrm{V}}(ML - n_{\mathrm{R}})\mathcal{L}(\widehat{\theta_{n_{\mathrm{R}}}}) + 2(1 + ML + \frac{1}{2}(n_{\mathrm{R}} - 1))$$
$$MDL(n_{\mathrm{R}}) = -n_{\mathrm{V}}(ML - n_{\mathrm{R}})\mathcal{L}(\widehat{\theta_{n_{\mathrm{R}}}}) + \frac{1}{2}(1 + ML + \frac{1}{2}(n_{\mathrm{R}} - 1))\ln(n_{\mathrm{V}}) \ ,$$

où ML est le nombre de points temporels après la première étape de réduction, n_{R} le nombre de sources et

$$(\widehat{\theta_{n_{\mathrm{R}}}}) = \ln\left(\frac{(\lambda_{n_{\mathrm{R}}+1}\cdots\lambda_{ML})^{\frac{1}{ML-n_{\mathrm{R}}}}}{\frac{1}{ML-n_{\mathrm{R}}}(\lambda_{n_{\mathrm{R}}+1}+\cdots+\lambda_{ML})}\right)$$

le logarithme du maximum de vraisemblances des paramètres du modèle avec λ_i la i^{ème} plus grande valeur propre issue de l'analyse en composantes principales. Le nombre de composantes est obtenu en minimisant un de ces critères par rapport à n_{R}.

L'ACI a été appliquée avec succès pour analyser des données d'IRMf en utilisant différentes stimulations [143, 184]. Cependant, l'ACI présente plusieurs inconvénients. Tout d'abord, une fois l'ACI réalisée, il est nécessaire d'apporter une connaissance supplémentaire afin de savoir si chaque composante est attribuable à du bruit ou du signal. L'identification en bruit ou signal n'est pas univoque : les oscillations cardiaques, respiratoires ou liées à la tâche apparaissent fréquemment dans plus d'une composante et peuvent apparaître mélangées dans la même composante [203]. De plus, il n'existe pas de notion d'ordre [232]. Il reste donc à ordonner les composantes selon le critère voulu (amplitude, position, fréquence, ...). Comparés à l'approche classique (analyse univariée par GLM), les résultats obtenus sont assez similaires, même si l'ACI se révèle parfois plus sensible. Comme toute approche fondée sur les données, l'avantage de l'ACI est de séparer les signaux temporels, sans faire d'hypothèse a priori sur leur décours. C'est pourquoi l'ACI est l'approche de référence pour analyser la connectivité fonctionnelle à partir de données d'IRMf au repos où l'absence de stimulation ne permet pas de définir un modèle [20]. L'ACI est aussi utilisée pour « débruiter » les données. L'idée est d'identifier les composantes indépendantes associées au bruit, le modèle mathématique de l'ACI permettant de reconstruire ensuite les données en enlevant ces composantes. Il s'agit donc d'une méthode de prétraitement dont l'efficacité repose sur la sélection des composantes d'origine neuronale. Cette approche est détaillée dans la section suivante.

2.2.2.3 Débruitage des données par l'ACI

L'ACI a aussi été développée pour tenter de débruiter le signal. L'idée est d'identifier les composantes indépendantes qui sont du bruit de celles qui représentent du signal puis de reconstruire le signal en ne gardant que les composantes indépendantes de signal. Il existe différentes méthodes pour classer les composantes selon des critères très différents [221]. Ne sont mentionnées ici que les approches utiles pour séparer bruit et signal en IRMf.

La classification par le pourcentage de variance expliquée n'apparaît pas comme idéale car les sources de bruit génèrent souvent plus de variance que les sources d'origine [232]. Différentes approches ont été proposées. Par exemple, Moritz *et al.* proposent d'ordonner les composantes indépendantes selon le spectre de puissance obtenu par analyse de Fourier [149]. Ces spectres de puissance ont été triés dans l'ordre décroissant de l'amplitude du pic correspondant à la fréquence fondamentale de la tâche. Cet ordonnancement combiné avec une inspection rétrospective des évolutions temporelles et des cartes spatiales permet de séparer le bruit des activations. Par exemple, dans le cas de mouvements de la tête corrélés à la tâche, l'ordonnancement par le spectre de puissance des composantes spatiales assigne un rang élevé à certaines composantes associées aux artéfacts de mouvements mais une inspection visuelle de leur carte spatiale permet de les identifier comme artéfacts. Zeng *et al.* proposent d'ordonner les composantes indépendantes par

la moyenne des maxima de corrélation (MMC *maximum mean correlation*) [232]. Ils se sont fondés sur un résultat de fiabilité de l'ACI obtenu par McKeow [144] montrant que réaliser une ACI sur les données ne comportant que les points temporels pairs ou que des points temporels impairs conduit à des composantes semblables à l'ACI réalisée sur l'ensemble des données, si la composante n'est pas du bruit. Pour chaque ensemble de données deux ACIs sont réalisées une sur tous les points temporels (composantes ALL) et l'autre sur les points temporels impairs (composantes ODD). Puis les corrélations spatiales (r_s) et temporelle (r_t) sont calculées entre chaque composante ALL et chaque composantes ODD où seul le maximum est retenu. Cette méthode reste inférieure à la méthode d'ordonnancement de Moritz mais elle ne nécessite pas de connaissance *a priori*. En outre, les auteurs ont trouvé que les composantes significatives ont habituellement une haute reproductibilité à la fois temporelle et spatiale alors que la plupart des composantes de bruit bien qu'elles aient une haute reproductibilité temporelle ont une reproductibilité spatiale plus basse.

Plutôt que d'utiliser des critères pour ordonner les composantes puis les inspecter manuellement, plusieurs auteurs ont tenté de développer des méthodes de classification automatique pour séparer les composantes assimilables à du bruit de celles assimilables à du signal. Thomas *et al.* séparent le bruit structuré (respiration et battements cardiaques) du bruit non-structuré en décomposant le signal soit par l'ACP soit par l'ACI [203]. Ils utilisent pour cela une analyse fréquentielle en identifiant le bruit structuré grâce aux fréquences cardiaque et respiratoire. Ils préfèrent utiliser l'ACI pour identifier les signaux cardiaques et respiratoires ; et l'ACP pour identifier le bruit non structuré. D'autres auteurs ont développé des classifieurs automatiques pour distinguer le signal du bruit [141, 172, 173, 207]. Pour cela, ils ont défini différents critères permettant de distinguer le bruit du signal. Ces critères peuvent se classer en trois grandes catégories : la première utilise la distribution spatiale des composantes indépendantes, la deuxième fait appel à l'analyse de Fourier et la dernière concerne le domaine temporel. La classification s'effectue alors en effectuant des combinaisons de tests logiques portant sur les différents critères. Tohka *et al.* utilisent six critères et distinguent quatre classes de bruit. De Martino *et al.* emploient onze critères dans [141] une SVM aux moindres carrés (*least square Support vector machine*), plutôt qu'un arbre de décision comme Tokha, et classent les composantes indépendantes en six catégories dont une spécifique au signal d'intérêt. Prium *et al.* se servent seulement de quatre critères et identifient trois classes : une pour le mouvement et une pour un réseau au repos [172, 173]. Ils ont montré que leur classifieur n'aurait pas besoin d'être entraîné à chaque nouveau jeu de données. Dans une étude comparative de leur méthode avec différentes stratégies de correction du mouvement, ils ont montré que leur méthode permet de réduire les effets significatifs du mouvement de la tête tout en préservant le signal d'intérêt.

Cependant, les classifieurs automatiques ont deux inconvénients majeurs. Tout classifieur nécessite une phase d'apprentissage qui consiste d'abord en la classification manuelle par un expert d'un jeu de données similaires à celui à analyser en différentes classes, suivie de la détermination des fonctions logiques discriminantes pour chaque classe (choix des critères et apprentissage du seuil). De plus, la plupart des classifieurs ont besoin d'être réentraînés lorsque le paradigme des nouveaux jeux de données s'éloigne de celui du jeu sur lequel ils ont été entraînés.

2.2.2.4 Analyse en composantes indépendantes et inférences de groupe

Un autre défi de l'analyse en composantes indépendantes est son extension à des analyses de groupe. Contrairement au modèle linéaire général où tous les sujets partagent le même modèle, l'ACI est difficilement généralisable au groupe car chaque individu a des composantes différentes temporellement et spatialement. Or, les ACIs de groupe sont utiles pour séparer bruit et signal car une caractéristique sous-jacente du signal d'intérêt est qu'il est commun pour la plupart des sujets. C'est pourquoi il est intéressant de pouvoir mettre en œuvres des ACIs exploratoires à l'échelle du groupe.

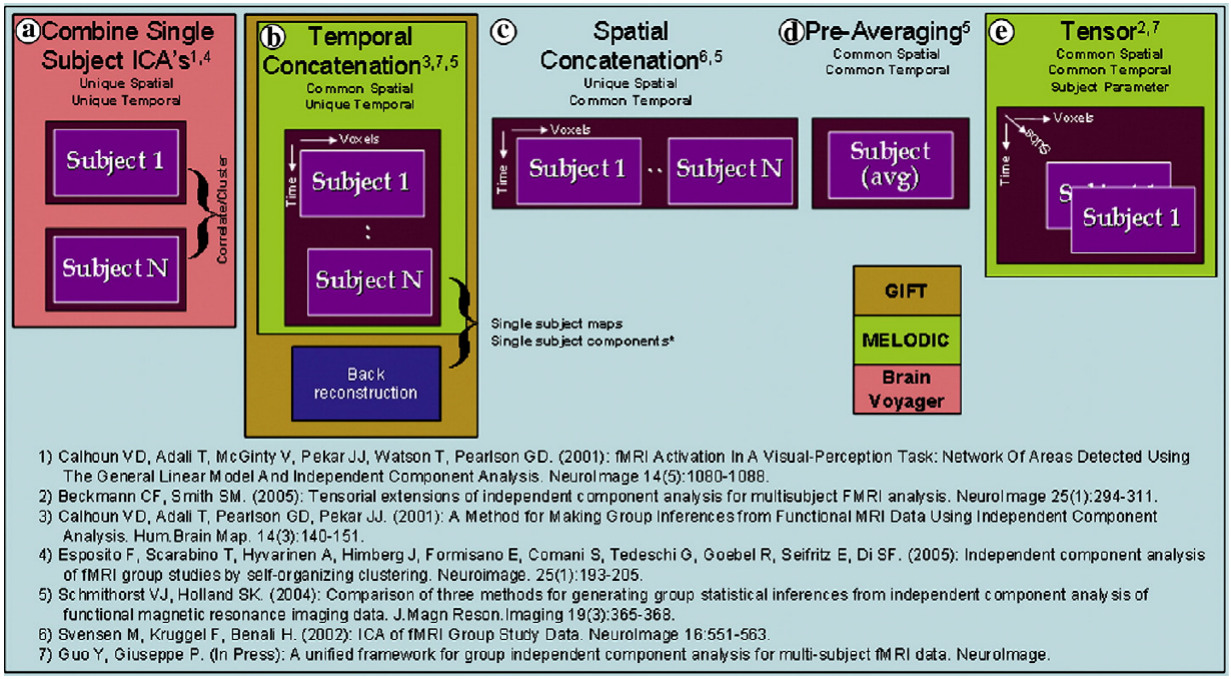


FIGURE 2.5 – Les différentes manières de faire une analyse en composantes indépendantes de groupe issue de [36].
a) ACI pour chque sujet séparément puis groupement par corrélation ou parcellisation ; b) Concaténation temporelle puis ACI sur les données concaténées ; c) Concaténation spatiale puis ACI sur les données concaténées ; d) ACI effectuées sur les données moyennées sur le groupe ; e) Approche par tenseur : les données sont représentées dans un cube pour représenter les dimensions spatiale, temporelle et sujet.

Calhoun *et al.* distinguent cinq catégories d'ACI de groupe présentées en figure 2.5 [36]. La première catégorie utilise des ACIs sur les sujets pris individuellement puis essaye de combiner les sorties en un groupe post hoc en utilisant des approches comme le *clustering* ou la corrélation spatiale des composantes ce qui tient compte des traits temporels et spatiaux uniques [66, 102]. Mais comme les données sont bruitées, il n'existe pas nécessairement des composantes communes au groupe. Par exemple, Esposito *et al.* proposent de calculer une similarité entre les composantes indépendantes des différents sujets à partir des coefficients de corrélation spatial et temporel. Leur idée est ensuite d'utiliser de parcelliser (autrement dit par *clustering*) la matrice de ces similarités afin de trouver les composantes appartenant à plusieurs sujets. Hyvärinen *et al.* mesurent quant à eux la similarité à partir des matrices $\boldsymbol{S}_k$ des différents sujets et s'en servent pour calculer des valeurs de p [102]. L'algorithme de parcellisation est ensuite appliqué sur ces valeurs de p, après correction pour test multiple, pour réunir les composantes qui sont assez similaires chez les différents sujets pour qu'on puisse les considérer comme une seule composante.

Les deux approches suivantes consistent à calculer l'ACI sur le groupe en concaténant les données soit spatialement [197] soit temporellement [33]. Leur avantage est de calculer une seule ACI qui peut ensuite être séparée entre les différents sujets ce qui simplifie la comparaison des sujets au sein d'une même composante. La concaténation temporelle tient compte de l'unicité des évolutions temporelles de chaque sujet mais suppose des cartes spatiales communes. À l'inverse, la concaténation spatiale tient compte de l'unicité des cartes spatiales mais suppose une évolution temporelle commune. L'un des avantages de la concaténation spatiale est qu'elle ne nécessite pas de normaliser les données dans un espace commun de référence. De plus, en concaténant spatialement, les sources ayant des caractéristiques temporelles différentes entre sujet (ce qui devrait inclure les sources de bruit) tendent à être supprimées. Schmithorst *et al.* ont montré que la concaténation temporelle semble mieux fonctionner pour les données d'IRMf [183].

Une autre approche implique de moyenner les données avant de réaliser l'ACI de groupe [182] : l'approche est moins coûteuse en calcul mais a une hypothèse plus forte de carte spatiale et évolution temporelle commune. La dernière approche est celle proposée par Beckmann *et al.* et fait appel à des tenseurs pour représenter à la fois l'évolution temporelle, les cartes spatiales et les modes individuels [19]. Cependant cette approche peut ne pas fonctionner correctement si les réponses temporelles des sujets sont très différentes [36].

Comme le soulignent Erhardt *et al.*, l'ACI de groupe est pertinente car elle est robuste et s'interprète naturellement [65]. Comme toute analyse de groupe, elle gomme une part de la variabilité inter-sujet selon la méthode choisie. D'un point de vue pratique, inférer par exemple que tous les sujets partagent la même réponse dans les mêmes territoires n'a pas la même signification que de présenter la même réponse dans des territoires différents. Il est très utile de pouvoir choisir la méthode d'inférence de groupe (type de concaténation par exemple) au regard de la question posée. À l'inverse, le principal avantage des ACI sujet par sujet est que les composantes trouvées révèlent les propriétés spatiotemporelles des réponses du sujet [66]. Elles autorisent alors une analyse détaillée de la variabilité inter-individuelle.

2.2.3 Corrélation inter-sujet (ISC)

Une autre méthode d'analyse conduite par les données est la corrélation inter-sujet (ISC) ou corrélation inter-participants. Introduite par Hasson *et al.*, cette analyse se fonde sur l'idée que pour une même stimulation, la stimulation active les mêmes régions et dans ces régions, la réponse est similaire [92]. Pour mesurer la similarité de la réponse temporelle, le coefficient de corrélation de Pearson est employé. L'idée de la corrélation inter-sujet est de calculer voxel par voxel et pour chaque paire de sujet, la corrélation entre les deux réponses temporelles pour lesquelles l'ordre de stimulation est le même. La figure 2.6 illustre le principe. Notons n_{S} le nombre de sujets et n_{V} le nombre de voxels considérés, notons $s^n(v,t)$ le signal dans le voxel v au temps $t \in [1, n_{\mathrm{T}}]$ en ordonnant les stimulations de manière identique pour tous les sujets. Comme les stimulations sont souvent présentées dans un ordre aléatoire, la réponse temporelle doit être réordonnée afin de corréler la réponse temporelle des différents sujets correspondant à la même stimulation. Notons $r^{m,n}(v)$ le coefficient de corrélation de Pearson entre le signal du sujet n et le signal du sujet m dans le $v^{ème}$ voxel qui s'écrit donc :

$$r^{n,m}(v) = \frac{\sum_{t=1}^{n_{\mathrm{T}}} \left(s^n(v,t) - \overline{s^n(v)}\right)\left(s^m(v,t) - \overline{s^m(v)}\right)}{\sqrt{\sum_{t=1}^{n_{\mathrm{T}}} \left(s^n(v,t) - \overline{s^n(v)}\right)^2 \sum_{t=1}^{n_{\mathrm{T}}} \left(s^m(v,t) - \overline{s^m(v)}\right)^2}}$$

La corrélation inter-sujet au voxel v consiste à moyenner les coefficients de corrélation $r^{m,n}(v)$ calculés pour chaque paire de sujets ce qui donne :

$$ISC(v) = \frac{2}{n_{\mathrm{S}}(n_{\mathrm{S}}-1)} \sum_{1 \leq m < n \leq n_{\mathrm{S}}} r^{m,n}(v) \ .$$

La corrélation inter-sujet est donc calculée en chaque voxel. Une carte spatiale contenant les corrélations entre tous les sujets est ainsi obtenue. Il faut ensuite pouvoir déterminer quelle corrélation est significativement importante. Le principe clé de cette approche est de détecter les territoires qui présentent des corrélations significatives. La méthode effectue donc une inférence de groupe en recherchant les mêmes réponses (au sens de la corrélation) dans les mêmes territoires Cependant, le coefficient de corrélation de Pearson ne suit pas une loi normale. Le coefficient de corrélation peut être converti en z-score par la transformation de Fisher : $z = \ln(\frac{1-r}{1+r})$ qui suit une loi normale [93, 224]. Une autre manière de faire est de calculer la distribution empirique de la corrélation inter-sujet à l'aide de permutations. Une fois la distribution empirique estimée, les valeurs de p peuvent être calculées pour une corrélation inter-sujet donnée le nombre de fois

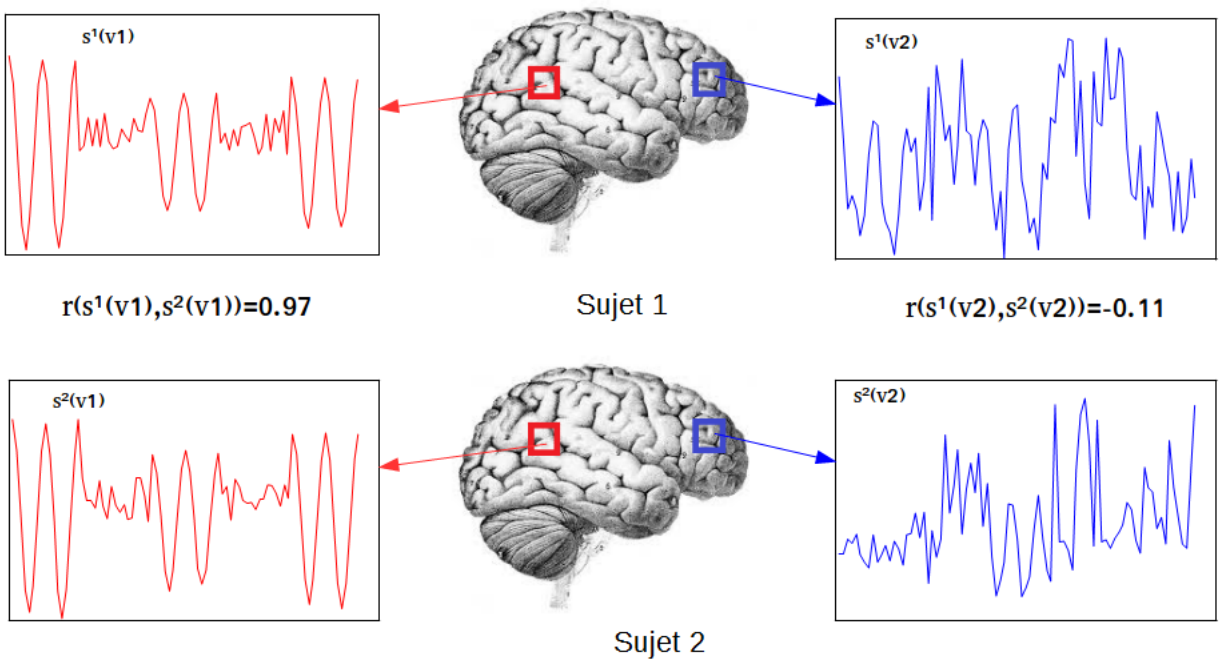


FIGURE 2.6 – Corrélation inter-sujet .
Les carrés représentent chacun un voxel différent. Sur la ligne du haut, sont montrés le signal mesuré dans chacun des voxels (la couleur du signal correspondant à la couleur du voxel) pour le premier sujet. La ligne du bas montre les signaux mesurés dans les deux mêmes voxels pour un deuxième sujet. Le coefficient de corrélation (r) est calculé entre les signaux des deux sujets pour chaque voxel. À gauche, les signaux sont fortement corrélés ce qui traduit que la réponse est similaire chez les deux sujets. À droite, les signaux sont faiblement corrélés : les réponses des deux sujets sont différentes entre elles.

où la distribution empirique donne une corrélation inter-sujet supérieure à celle évaluée. Tokha explique qu'il s'agit de tester s'il y a une synchronisation de l'activité neuronale entre les sujets dans le voxel considéré, ce qui est différent de tester si la corrélation inter-sujet est nulle [206]. Le problème est que pour être valide, ce test non-paramétrique doit tenir compte des dépendances suivantes : les corrélations temporelles dans le signal d'IRMf, l'utilisation du signal d'un sujet $n_S - 1$ fois (donc les coefficients de corrélation ne sont pas indépendants) et la dépendance spatiale entre le signal temporel des voxels voisins. Tohka propose donc d'obtenir la distribution en faisant une permutation circulaire du signal c'est-à-dire qu'on tire aléatoirement un temps $t_n \in [1, n_T]$ et qu'au lieu de prendre pour le sujet n le signal dans l'ordre naturel, on calcule la corrélation inter-sujet avec le signal $[s^n(v, t_n + 1), s^n(v, t_n + 2), \ldots, s^n(v, n_T), s^n(v, 1), \ldots, s^n(v, t_n)]$ [206]. L'idée de la procédure (utilisée dans l'ISCToolbox [115]) est de choisir aléatoirement un voxel, de tirer aléatoirement n_S entiers $(t_n)_{1 \leq n \leq n_S}$ et de calculer la corrélation inter-sujet sur les signaux permutés ; puis de répéter la procédure un grand nombre de fois. Les corrélations inter-sujets obtenues forment alors la distribution empirique permettant de calculer les valeurs de p. Pajula *et al.* ont montré que les résultats obtenus avec l'ISC sont fiables à partir d'un groupe de vingts sujets [163]. Chen *et al.* ont critiqué l'approche implémentée dans l'ISCToolbox [42]. Tout d'abord, ils considèrent que moyenner les coefficients de corrélation $r^{m,n}$ pose problème car le coefficient de corrélation n'est pas une fonction linéaire de la force de la relation entre deux variables. Moyenner les coefficients de corrélation sous-estime l'effet de groupe car l'estimation tend vers 0. Par exemple, si on prend deux coefficients de corrélation $r = 0,5$ et $r = 0,9$, la moyenne donne $0,7$ alors qu'en passant par la transformation de Fisher, une corrélation de 0,77 est obtenue. Ils ont également comparé la méthode de permutation de Tokha avec une permutation sujet par sujet ou élément par élément et un bootstrapping sujet par sujet ou élément par élément [42]. Ils ont montré que dans le cas d'un groupe, la méthode par bootstrapping sujet par sujet contrôle

mieux le taux de faux positif que les autres méthodes [42].

La ISC a ensuite été étendue à la comparaison de groupe [42, 91],à l'analyse individuelle en calculant la corrélation intra-sujet c'est-à-dire la corrélation entre les différentes répétitions d'une même stimulation [78, 93] et au domaine fréquentiel [114] en calculant la corrélation intersujet après une décomposition en ondelettes. Kauppi *et al.* ont également proposé de comparer deux conditions en regardant s'il y a des différences significatives entres les ISCs calculées sur deux conditions différentes.

Comme toute méthode guidée par les données, la méthode de corrélation inter-sujet permet d'identifier les aires cérébrales activées sans avoir besoin ni d'une matrice de dessein (paradigme), ni de modèle de la réponse fonctionnelle. En revanche, elle ne permet pas d'analyse individuelle. Elle a été très utilisée pour l'analyse de données d'IRMf obtenues avec une stimulation naturelle comme le visionnage d'un film [78, 92, 93, 110, 98, 114, 157], la synchronisation des émotions [157], la perception d'un dilemme moral [15], l'écoute d'un récit [224, 123] ou de musique [4, 126]. Pour trouver quels moments de la stimulation suscite la synchronisation entre les sujets, Hasson *et al.* proposent d'utiliser la corrélation inversée [92] : en se plaçant dans une région où la corrélation inter-sujet est significative, ils moyennent le signal temporel sur les sujets et identifient à partir des pics de ce signal moyen les moments de la stimulation suscitant la synchronisation entre les sujets. Cela étant, le niveau de corrélation dépend beaucoup du niveau de bruit présent dans les données. Pajula *et al.* montrent qu'il est nécessaire de filtrer afin d'assurer un ratio signal sur bruit suffisant et de limiter l'impact des erreurs de normalisation [162].

2.2.4 Parcellisation (ou *clustering*)

Parcelliser le cerveau consiste à regrouper les voxels en régions distinctes, ces régions étant homogènes selon un critère donné. Nous décrivons ici les principales techniques de parcellisation.

Notons par la suite $\boldsymbol{x}_v$ le signal temporel du $v^{ème}$ voxel de taille n_T et notons n_V le nombre de voxels.

2.2.4.1 *k-means*

Soit n_C le nombre de parcelles à trouver en deux étapes. L'algorithme des k-moyennes consiste à choisir n_C voxels parmi les n_V, comme parcelles initiales. Notons C_i la $i^{ème}$ parcelle et $\overline{C_i}$ son centroïde défini comme la moyenne des signaux temporel de chaque voxel dans la parcelle C_i. Pour chaque voxel, la similarité entre $\boldsymbol{x}_v$ et chacun des n_C centroïdes. Chaque voxel est ensuite attribué à la parcelle pour laquelle la similarité est maximale. Pour les n_C parcelles, leur centroïde est recalculé à partir des voxels dans la parcelle. Les étapes d'attribution des voxels dans les parcelles et de mise à jour des centroïdes sont répétées jusqu'à ce qu'un critère de stabilité soit atteint (les parcelles n'évoluent plus par exemple). Le problème de l'algorithme *K-means* est que les parcelles finales dépendent de l'initialisation des n_C parcelles initiales donc deux exécutions sur le même jeu de données peuvent donner des parcellisations différentes. De plus, le résultat dépend également du critère de similarité choisi.

Kahnt *et al.* emploient l'algorithme *K-means* avec la corrélation comme similarité pour parcelliser le cortex orbito-frontal (**cf. section A.2**) [111]. Similairement, Yeo *et al.* ont parcellisé le cortex en fonction de la connectivité fonctionnelle et ont ainsi trouvé des réseaux d'aires primaires adjacentes et des réseaux constitués d'aires éparpillées dans le cortex [205].

Golland *et al.* ont utilisé l'algorithme *K-means* sur des données d'IRMf issues de plusieurs expériences [79] et ont divisé le cerveau en deux réseaux : un réseau intrinsèque s'apparentant au réseau de mode par défaut et un plus extrinsèque. Gonzalez-Castillo et al. utilisent également l'algorithme des k-moyennes dans [81] pour parcelliser à l'échelle individuelle le signal moyenné sur 100 répétitions d'un paradigme moteur en choisissant comme similarité le coefficient de corrélation de Pearson. Ils ont obtenu des parcellisations symétriques, comparables entre sujets

et correctes en termes d'anatomie fonctionnelle. En outre, plusieurs parcelles correspondant à des régions répondant à la stimulation ont été identifiées.

2.2.4.2 Hierarchical agglomerative clustering (HAC)

Les algorithmes de clustering hiérarchiques agglomératifs (HAC) commencent par considérer chaque voxel comme une parcelle comportant un unique élément. À chaque itération, une paire de parcelles est fusionnée en une parcelle regroupant les deux parcelles selon un critère de similarité. Cela produit donc une hiérarchie de parcelles aussi appelé dendrogramme. Il existe plusieurs algorithmes hiérarchiques agglomératifs qui se distinguent selon deux éléments : la similarité utilisée entre les voxels et le critère d'agglomération qui détermine de quelle manière les nouveaux membres sont ajoutés aux parcelles.

Michel *et al.* ont développé un clustering hiérarchique agglomératif (HAC) avec le critère de Ward et contraint spatialement (seuls les voxels voisins peuvent être dans la même parcelle) prenant en compte la structure spatiale et l'information multivariée des données [146]. L'originalité de leur approche est qu'ils utilisent un algorithme d'apprentissage pour déterminer la meilleure répartition des voxels dans les parcelles (en fonction du score d'apprentissage). Leur approche permet de se concentrer sur de petites parcelles informatives et laisse les aires non-informatives non parcellisées. Wang *et al.* parcellisent des données d'IRM au repos à l'aide d'un clustering hiérarchique agglomératif avec le critère du lien moyen (qui calcule la similarité entres parcelles comme la moyenne des similarités entre les voxels d'une paire appartenant respectivement à l'une ou l'autre des deux parcelles) et le coefficient de corrélation comme distance entre les voxels [220]. Ils se limitent à des voxels dans la substance grise et seuillent la matrice de corrélation croisée mettant à 0 toutes les corrélations inférieures à 0.3. L'originalité de leur approche est qu'ils utilisent la taille des parcelles comme critère pour raffiner la parcellisation : si une parcelle comporte plus de 5 000 voxels, elle est divisée en sous-parcelles. Parmi les 37 parcelles trouvées, ils en ont identifiées 20 qui correspondent à des réseaux observés au repos.

Orban *et al.* emploient un HAC avec le critère de Ward et la distance euclidienne afin de parcelliser des données issues d'une expérience de contrôle moteur [160]. Leur approche est multi-niveau et utilise des procédures de bootstrapping. Le premier niveau est le niveau individuel, ils utilisent le signal moyen dans chacune des 957 régions et le moyennent sur les répétitions. Ils parcellisent alors les signaux ainsi obtenus afin d'obtenir une matrice de stabilité, qui consiste à mesurer le nombre de fois où les régions sont assignées à la même parcelle pour une échelle K donnée. Ils répètent la procédure plusieurs fois en tirant avec remise les répétitions. Les matrices de stabilité individuelles sont ensuite parcellisées ce qui donne une matrice de stabilité de groupe pour une échelle L donnée. Ils répètent l'expérience plusieurs fois en tirant avec remise les sujets et ils moyennent ensuite les matrices de stabilité de groupe ainsi obtenue. Cette matrice de stabilité moyenne de groupe est ensuite parcellisée à une échelle M donnée. Finalement, ils arrivent à décomposer le cerveau en des réseaux liés à la tâche stables à l'échelle du groupe. Gonzales-Castillo *et al.* ont trouvé des résultats similaires à ceux obtenus par l'algorithme *K-means* en utilisant le HAC avec le critère de Ward et la distance euclidienne [81], ce qui conforte leurs résultats puisqu'ils sont retrouvables à partir de deux méthodes de parcellisation différentes.

2.2.4.3 *Spectral clustering*

Le clustering spectral consiste en exécuter un algorithme *K-means* non sur les données directement mais sur une transformation des données qui permet de préserver la structure spatiale tout en représentant la similarité fonctionnelle. Soit pour chaque paire de voxels, $\boldsymbol{W}$ représente alors une matrice d'adjacence (autrement dit de voisinage) pondérée par la distance fonctionnelle choisie. Soit alors $\boldsymbol{\Delta_W}$ la matrice diagonale contenant la somme des lignes de $\boldsymbol{W}$. Le clustering spectral consiste alors à appliquer l'algorithme *K-means* sur les m premières solutions de l'équation $\boldsymbol{W}\xi = \lambda\boldsymbol{\Delta_W}\xi$ [202].

Plusieurs auteurs ont utilisé le *clustering* spectral pour parcelliser les données d'IRMf [201, 50, 107]. Ce qui change entre les différents algorithmes proposés est la définition de la matrice d'adjacence $\boldsymbol{W}$ et la coupure du dendrogramme pour obtenir la parcellisation finale. Afin de parcelliser sur un critère fonctionnel et anatomique, Thirion *et al.* construisent leur matrice d'adjacence à l'aide des β obtenus à l'aide d'un GLM, et imposent des contraintes spatiales (seuls les voxels voisins spatialement peuvent être regroupés) [201]. Craddock *et al.* et James *et al.* emploient le coefficient de corrélation comme matrice d'adjacence [50, 107].

Plutôt que de proposer uniquement une parcellisation fondée sur des critères anatomiques (grâce à la distance géodésique) et sur l'algorithme *K-means* comme Flandin *et al.*[69], Thirion *et al.* utilisent le *clustering* spectral pour parcelliser le cerveau en régions anatomiques et fonctionnelles. Pour cela, ils construisent leur matrice d'adjacence à l'aide des β obtenus à l'aide d'un GLM, en imposant que seuls les voxels voisins puissent être regroupés dans une même parcelle. En calculant les contrastes dans les parcelles, ils obtiennent des activations similaires à celles après inférence univariée mais avec une plus grande sensibilité.

2.2.4.4 Méthodes de parcellisation de groupe

Plusieurs approches ont été développées pour créer une parcellisation de groupe. Ces approches peuvent se diviser en deux catégories :

1. appliquer un algorithme de parcellisation sur chaque sujet individuellement puis appliquer un deuxième algorithme de parcellisation à ces parcellisations individuelles ;
2. calculer une matrice représentant les caractéristiques à l'échelle de groupe (par concaténation de la réponse temporelle par exemple) et appliquer un algorithme de parcellisation à cette matrice [12].

Les travaux de van den Heuvel *et al.* et ceux de Arslan *et al.* s'inscrivent dans la première approche [212, 13]. En effet, van den Heuvel *et al.* proposent de parcelliser à l'échelle du groupe une matrice qui pour chaque paire de voxels compte le nombre de parcellisation individuelle les classant dans la même parcelle [212]. Arslan *et al.* reprennent la même technique mais les parcellisations individuelles sont elles-mêmes obtenant après une première parcellisation individuelle combinant critère anatomique et fonctionnelle [13]. Parmi les travaux utilisant la deuxième approche, nous pouvons citer les travaux de Thirion *et al.* qui concatènent les données des différents sujets en un seul jeu de données pour ensuite appliquer sur ce jeu de données un algorithme de parcellisation [202, 201] ; et le moyennage de la distance sur l'ensemble des sujets proposée par Patel *et al.* [178].

2.2.5 *Temporal clustering analysis*

L'analyse par clustering temporel (ou *Temporal clustering analysis* (TCA)) est une autre méthode conduite par les données que nous ne classons pas dans les méthodes de parcellisation vues précédemment. En effet, cette technique n'a pas pour but de regrouper les voxels mais tendrait à regrouper les points temporels sans pour autant définir de similarité entre les points temporels. La TCA est fondée sur l'hypothèse que le nombre de voxels avec des valeurs maximales est plus grand pendant les phases d'activations que pendant les phases de repos.

Considérons la matrice $\boldsymbol{X}^\top$ de taille n_V par n_T dont la ligne v contient le signal mesuré dans le voxel v. Définissons maintenant la matrice $\boldsymbol{W}$ de taille n_V par n_T par :

$$w_{vt} = \begin{cases} x_{tv} \text{ si } t = \arg\max\{x_{pv}, p \in \{1, \ldots, n_\mathrm{T}\}\} \\ 0 \text{ sinon} \end{cases} . \tag{2.1}$$

L'équation 2.1 définit un critère qui indique si le signal mesuré dans le voxel v atteint son intensité maximale. Est ensuite défini le vecteur $\boldsymbol{k}$ correspondant à l'évolution du nombre de

voxels dont la variation de signal est maximale au cours de temps en sommant les éléments de chaque colonne de la matrice $\boldsymbol{W}$:

$$\boldsymbol{k} = \left(\sum_{t=1}^{n_\mathrm{T}} w_{1t}, \ldots, \sum_{t=1}^{n_\mathrm{T}} w_{n_\mathrm{V}t}\right) \quad .$$

Le vecteur $\boldsymbol{k}$ représente l'évolution au cours de temps du nombre de voxels dont l'intensité de signal est maximale. Il peut donc être utilisé pour détecter les périodes d'activations sans avoir de modèle de la réponse temporelle. Pour cela, les vecteurs $\boldsymbol{k}$ sont calculés pour chaque sujet puis moyennés sur le groupe. Les voxels actifs sont ensuite identifiés en comparant la moyenne du signal pendant ces phases d'activation et la moyenne du signal pendant la phase de repos (les points temporels ne correspondant pas à une période d'activation).

La TCA a été appliquée avec succès pour détecter les changements de la réponse BOLD après l'ingestion d'une boisson à base de glucose [131]. Elle a ensuite été améliorée pour augmenter sa sensibilité [229] et pour détecter plusieurs phases d'activations [74] par une procédure itérative. Elle a également été généralisée à l'ensemble du cerveau [136]. Morgan *et al.* ont réussi à détecter l'activité épileptique chez des patients épileptiques avec la TCA [148]. Cependant, leurs résultats ont été réfutes par Hamandi *et al.* qui ont montré que le vecteur $\boldsymbol{k}$ est corrélé aux paramètres de mouvements [88].

2.3 Quelle méthode d'analyse conduite par les données choisir ?

À notre connaissance, il n'existe aucune comparaison exhaustive des différentes méthodes d'analyse guidée par les données. Il semble pourtant que l'analyse en composantes indépendantes soit la plus répandue, grâce au développement de GiFT développé par Calhoun *et al.* [34]. C'est pourquoi l'ACI a été utilisée non seulement pour détecter les activations mais également pour débruiter les données comme montré en section 2.2.2.3. Correa *et al.* ont par ailleurs comparé cinq algorithmes implémentés dans GIFT et ont montré que l'algorithme Infomax est le meilleur choix pour les données d'IRMf car il permet de distinguer les différents sortes de sources en ayant un un meilleur ratio contraste sur bruit [48]. Les autres méthodes guidées par les données ont ainsi été comparées soit à l'ACI soit au GLM suivi d'une inférence univariée afin de montrer leur validité. Pajula *et al.* ont ainsi comparé l'ISC et le GLM à l'aide du coefficient de corrélation de Pearson spatial et de l'indice de Dice (sur les cartes d'activation binarisées) [161]. Ils ont ainsi montré que les activations détectées par l'ISC correspondent bien aux activations détectées par le GLM.Wilson *et al.* ont aussi comparé le GLM avec l'ISC lors d'une compréhension orale d'un récit [224]. Quelle que soit la méthode employée, ils trouvent des régions impliquées dans la compréhension du discours. Mais, ils ont également montré que l'ISC révèle des zones cérébrales rarement rapportées pour la compréhension écrite d'un récit, bien qu'elles aient été rapportées lors de tâches de compréhension écrite. Cependant, malgré le développement d'un logiciel implémentant l'ISC [115], la corrélation inter-sujet est restée confinée à quelques laboratoires et pour traiter des simulations naturelles. Un reproche adressé à l'ISC par Chen *et al.* dans [42] est le moyennage des coefficients de corrélation sans passer par une transformation de Fisher. De plus, contrairement au GLM ou à l'ACI, il n'existe pas de méthode assurant à la fois l'analyse individuelle et l'analyse de groupe. Concernant la *temporal clustering analysis*, elle a été comparée à l'ACI par Zhao *et al.* [234]. Ils ont montré que lorsque le contraste sur bruit est assez élevé, les deux méthodes donnent des résultats similaires que ce soit sur des données simulées ou réelles. Malgré sa simplicité, elle ne semble pas s'être répandue sans doute car il n'existe pas de méthode d'analyse de groupe. Concernant la parcellisation, Abraham *et al.* ont comparé l'ACI, l'ACP, l'algorithme *K-means* et le HAC avec le critère d'agglomération de Ward [2] et ont montré que l'ACP et l'ACI agrègent des réseaux fonctionnels capturant mieux la variance du signal que ceux trouvés par les méthodes de parcellisation. Mais les réseaux fonctionnels trouvés par la parcellisation sont plus stables sur l'ensemble des sujets que ceux trouvés par l'ACI et l'ACP.

Bien que l'ACI se soit imposée comme la méthode conduite par les données de référence, elle n'est pas excepte de critiques. Ainsi, Daubechies *et al.* ont mis en question le critère d'indépendance. Ils ont en effet montré que l'ACI spatiale avec les algorithmes InfoMax et FastICA peut capturer avec précision la variabilité spatiale même à une fine échelle mais que cela dépend plus de la sparsité des composantes que de l'indépendance des composantes trouvés [55]. Gao *et al.* ont remis en question l'utilisation quasi systématique de l'ACI spatiale et ont démontré que l'ACI spatiale et l'ACI temporelle ne sont pas totalement équivalentes [75]. Ces dernières années, la parcellisation semble remplacer l'analyse en composantes indépendantes en particulier pour les données d'IRMf au repos notamment pour l'étude de la connectivité fonctionnelle. Plusieurs comparaisons des algorithmes de parcellisation ont été effectué. Thirion *et al.* ont comparé quatre algorithmes de parcellisation appliqués à des images contrastes (obtenues après un GLM) : l'algorithme *K-means*, le HAC avec le critère de Ward avec des contraintes spatiales, le clustering spectral et l'algorithme *K-means* appliqué aux coordonnées spatiales [202]. Ils ont montré que le clustering spectral donne une parcellisation géométrique et donc tient moins compte de l'information fonctionnelle ; l'algorithme *K-means* présente des clusters disjoints spatialement mais plausibles fonctionnellement et l'algorithme de Ward (c'est-à-dire un algorithme HAC utilisant le critère d'agglomération de Ward) est un compromis entre les deux. Pour des échelles fines, ils recommandent d'utiliser l'algorithme de Ward avec contraintes spatiales. Arslan *et al.* ont d'ailleurs récemment étendu la comparaison faite par Thirion à un grand nombre d'algorithmes de parcellisation individuelle et de groupe [12] appliquées à des données acquises au repos. Ils définissent pour cela quatre catégories de critères : la reproductibilité entre sujets ou entre différentes acquisitions pour un même sujet ; l'homogénéité qui évalue la similarité du signal des voxels dans chaque parcelle ; la comparaison avec des parcellisations avec des cartes issues d'autres modalités (comme les aires de Brodmann) ; et la capacité à capturer la variabilité inter-individuelle et à expliquer les processus cognitifs observés. Ils ont montré que par rapport à l'ensemble des critères considérés, il n'existe pas un algorithme ou un type d'algorithme qui fasse consensus. Par exemple,l'algorithme *K-means* ou le HAC avec le critère de Ward et des contraintes spatiales donnent des parcelles homogènes mais sont peu reproductibles et se recoupent peu avec les autres modalités.

Néanmoins, en cohérence avec Thirion [202], l'algorithme de Ward avec contraintes spatiales apparaît comme un bon compromis entre l'algorithme des k-moyennes et le clustering spectral. Grâce à cette parcellisation, des réponses BOLD avec une riche variété de formes ont été observées dans l'ensemble du cerveau après l'exécution d'une stimulation visuelle [81] ou d'une stimulation motrice [160]. **L'algorithme de clustering hiérarchique avec le critère de Ward apparaît comme une alternative crédible à la GLM.** Cependant, un frein à la parcellisation des données d'IRMf est la quantité de données à traiter qui joue à la fois sur l'espace de stockage nécessaire mais également sur le temps d'exécution des algorithmes de parcellisation. Pour limiter ce problème, Orban *et al.* ont parcellisé le signal moyenné dans des régions prédéfinies à l'avance [160] et Gonzales Castillo *et al.* [81] ont réduit le nombre de voxels considérés. Thirion *et al.* ont proposé pour réduire la taille des données à parcelliser, d'ajouter des contraintes spatiales en ne regroupant que des parcelles voisines spatialement [202]. Cependant, il existe différents exemples de régions fonctionnellement connectées mais disséminées dans tout le cerveau [47].

L'idée de ce travail est donc d'utiliser l'algorithme de clustering hiérarchique avec le critère de Ward pour analyser les données d'IRMf fondées sur des tâches alimentaires. L'objectif est bien sûr de s'affranchir autant que possible de l'étape de modélisation qui peut gommer des différences subtiles entre réponses. De plus, s'affranchir de toute contrainte spatiale nécessite d'optimiser la mise en œuvre des algorithmes pour traiter les données dans des temps raisonnables (en quelques heures). Pour cela, il faut utiliser du calcul parallèle dont les principes sont décrits au chapitre 3.

Chapitre 3

Introduction au calcul haute performance

Ce chapitre définit ce qu'est le parallélisme et pourquoi il est devenu nécessaire. On s'intéresse ensuite à l'architecture d'un ordinateur et aux algorithmes dit *cache-aware* et les algorithmes dit *cache-oblivious*. Le chapitre se poursuit par une description des différentes architectures d'ordinateurs parallèles puis s'achève par la description des différents modèles de programmation correspondant.

3.1 Qu'est-ce que le parallélisme ?

Le parallélisme se définit comme l'ensemble des techniques logicielles et matérielles permettant l'exécution simultanée de séquences d'instructions indépendantes sur des processeurs ou cœurs différents. Les techniques matérielles correspondent aux différentes architectures de calculateur parallèle. Les techniques logicielles correspondent aux différents modèles de programmation parallèle [84].

Le parallélisme a plusieurs objectifs : soit l'accélération d'un programme (donc la réduction de son temps d'exécution) en distribuant le travail, soit le traitement de problèmes de plus grandes tailles en utilisant plus de ressources matérielles notamment la mémoire. Le parallélisme peut également servir dans des applications temps-critique ou pour minimiser les coûts de calcul [8].

De plus en plus d'applications de calcul scientifique nécessitent l'emploi du parallélisme que ce soit à cause de contraintes de temps : prédiction du climat à l'échelle mondiale, à cause de contraintes de coût : simulateur de crash d'avion ou de voiture, ou à cause de contrainte d'échelle : modélisation du climat, l'astrophysique ou la combustion à une échelle fine. Par exemple, l'exécution du code pour l'allumage d'une chambre de combustion d'hélicoptère prend 78 h sur 112 processeurs Intel Xeon hexa-cœurs cadencés à 2,6 Ghz contre près d'un an de calcul sur un seul processeur de même type [84]. De plus, les performances avec un seul processeur sont proches des limites physiques. En effet, une des manières d'accélérer la vitesse de calcul est d'accélérer la fréquence d'horloge. Cependant, la puissance électrique et la dissipation de la chaleur sont proportionnelles à la fréquence au cube, ce qui est limitant pour l'augmentation de la fréquence. Une deuxième manière est de faire du parallélisme d'instructions au niveau du matériel. Mais le gain en exécution concurrente est faible à cause des schémas d'accès mémoire irréguliers, de la dépendance des accès aux données et des calculs dépendants d'un contrôle. Une autre limitation d'un uni-processeur est la latence mémoire. En effet, l'écart s'est creusé entre la vitesse des processeurs et la vitesse d'accès mémoire. Ainsi pour délivrer une performance multipliée par deux avec la même bande passante, le taux de défaut de cache (qui correspond à la proportion d'accès à des données non présentes dans la mémoire cache) doit être réduit de moitié, ce qui signifie d'augmenter la taille du cache (par exemple d'un facteur quatre pour la multiplication matricielle) [30].

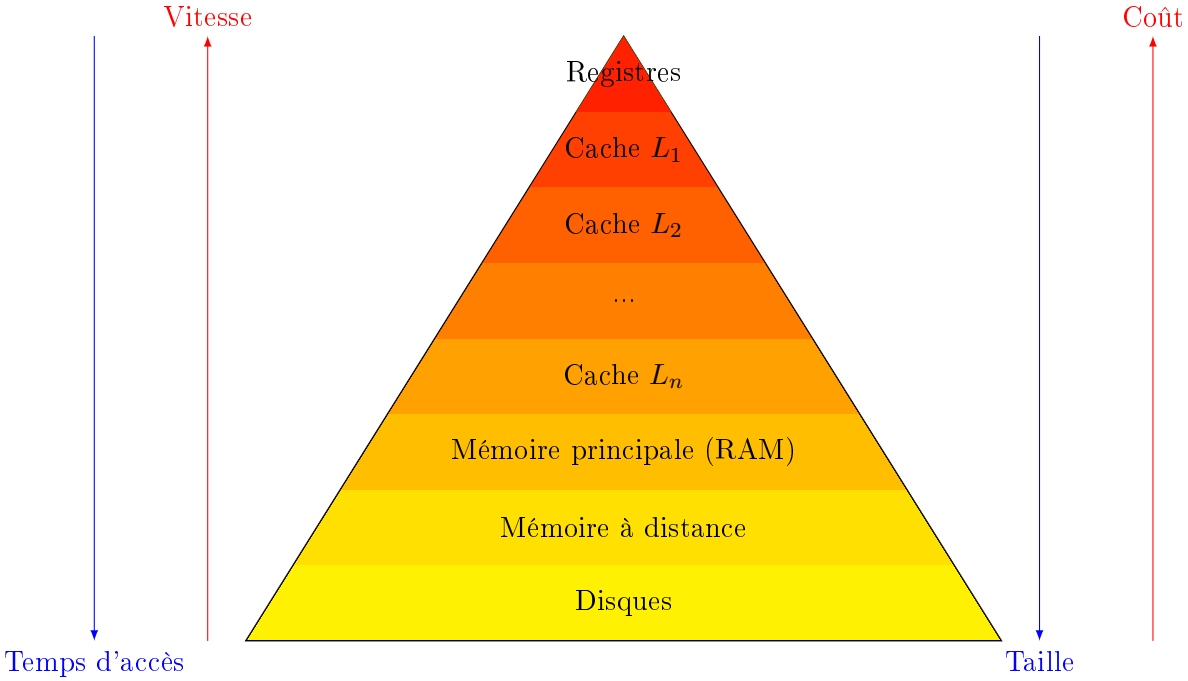


FIGURE 3.1 – Hiérarchie de la mémoire .

3.2 Architecture d'un ordinateur

Avant de s'intéresser aux systèmes parallèles, nous détaillons la hiérarchie de la mémoire d'un processeur et définissons la localité des données et les algorithmes *cache-aware* et *cache-oblivious*.

3.2.1 Terminologie

Un **processeur** ou *Central Processor Unit* (CPU) est le composant de l'ordinateur qui exécute les instructions qui lui sont données par le système d'exploitation [171]. Un **processus** est un programme en cours d'exécution par un ordinateur. Un processus peut avoir plusieurs processus légers qui se partagent la mémoire virtuelle du processus et peuvent donc collaborer sur les mêmes données [60].

3.2.2 Hiérarchie de la mémoire

Il existe une hiérarchie des mémoires informatiques : les mémoires les plus rapides sont placées près du processeur et les mémoires lentes sont placées loin du processeur. Plus la mémoire est rapide, plus elle est coûteuse à fabriquer et donc plus elle est petite.

La figure 3.1 montre la hiérarchie de la mémoire. La taille de la mémoire est exprimée en octets (un octet valant huit bits). On appelle kibioctet (noté Kio) $2^{10} = 1024$ octets, mébioctet (noté Mio) 2^{20} octets et gibioctet (noté Gio) 2^{30} octets. La latence est définie comme le temps passé entre une requête et le début de la réponse correspondante. En général, elle correspond au temps d'accès des données. Elle est exprimée en nombre de cycles. Les registres sont des mémoires intégrées au processeur d'accès très rapide en général 1 cycle mais de quelques dizaines d'octets. Les caches sont des mémoires intermédiaires entre le processeur et la mémoire centrale qui sont hiérarchisées. Le cache L_1 est accédé en quelques cycles (ce qui correspond à une vitesse maximale de 700 Gio/s) et fait environ 128 Kio. Le cache L_2 a une latence entre 2 et 10 fois supérieure au cache L_1 (vitesse maximale d'environ 200 Gio/s) pour une taille d'environ 1 Mio. Le cache L_3 fait environ 6 Mio mais a une vitesse de 40 Gio/s. La mémoire centrale fait entre une dizaine et une centaine de Gio mais a une latence de quelques centaines de cycles. Le disque

a une taille en général de plusieurs centaines de Gio voire plusieurs Tio, mais sa latence est de l'ordre de plusieurs millions de cycles.

La façon d'accéder aux données peut dégrader la performance d'un programme. Pour améliorer la performance, il faut limiter les défauts de cache et mieux utiliser la localité des données. Revenons d'abord à la mémoire cache. Le cache est divisé en lignes de mots. Lorsque le processeur doit accéder à une donnée deux cas de figure peuvent se produire : soit la donnée est présente dans la mémoire cache (*cache hit*) et le transfert vers les registres a lieu immédiatement ; soit la donnée ne se trouve pas dans le cache et c'est ce qu'on appelle un défaut de cache (ou *cache miss*) : il faut alors charger une nouvelle ligne dans le cache depuis la mémoire centrale et le processeur attend donc la fin du chargement de cette nouvelle ligne de cache. Pour charger cette nouvelle ligne de cache, il faut renvoyer en mémoire centrale une ligne déjà présente en cache, en général celle qui n'a pas été utilisée depuis longtemps. Il existe trois défauts de cache : le défaut de cache obligatoire qui survient la première fois que le programme fait référence à une donnée, qui peut être évité si le système réussit à prédire quelle sera la donnée suivante ; le défaut de cache dû à la taille du cache qui survient quand une donnée doit être écrasée car la taille du cache n'est pas suffisante pour contenir toutes les données utilisées par le programme et le défaut de conflit qui surviennent lorsque deux données sont associées au même endroit dans le cache. Pour éviter les défauts de cache dus à la taille mémoire, il faut partitionner les données en blocs suffisamment petits pour que les données restent dans la mémoire cache pendant assez de temps, ce qui suppose que les données soient utilisées plusieurs fois (une donnée utilisée une seule fois n'a pas besoin de rester dans la mémoire cache) [60].

3.2.3 Localité des données et algorithmes *cache-aware*

Pour éviter les défauts de cache, il faut utiliser le principe de localité des données. La localité temporelle des données est le principe que si un processus accède à une donnée, il est probable qu'il y accédera à nouveau dans un futur proche. La localité spatiale des données revient à l'idée que si un processus accède à une donnée, il accédera aux données voisines dans un futur proche. Comme le chargement des données se fait par ligne de cache, il est intéressant de programmer de manière à utiliser toutes les données de la ligne de cache, notamment lorsqu'on travaille sur des matrices et des vecteurs.

3.2.3.1 Algorithme *cache-aware*

Lorsque les applications sont limitées par la bande-passante ou par la latence, le temps d'accès mémoire ne peut alors se réduire qu'en réduisant le nombre de défauts de cache. En général, les temps de calculs sont plus rapides que les temps d'accès mémoire. Pour réduire les temps d'accès mémoire, il faut mieux exploiter la localité des données et notamment utiliser le modèle dit *cache-aware*. Dans ce modèle illustré en figure 3.2, on suppose qu'il existe deux niveaux de mémoire : le cache et la mémoire centrale. Le transfert des données entre la mémoire centrale et le cache s'effectue par bloc d'une taille connue B. Le cache a une taille mémoire fixée M et peut donc contenir $\frac{M}{B}$ blocs. Créer un algorithme *cache-aware* consiste donc à réarranger les accès mémoire pour tenir compte de la taille des blocs et de la taille du cache pour utiliser le cache de manière optimale.

Un réarrangement possible de l'algorithme est le *loop tiling* qui consiste à diviser une boucle en deux boucles imbriquées : la boucle externe itère sur des blocs et la boucle interne parcourt le bloc. Par exemple, le parcours d'une seule dimension montré en algorithme 1 devient le parcours 2. L'algorithme 3 montre qu'on parcourt toutes les cases d'un tableau (boucle ligne 2) plusieurs fois (boucle ligne 1) mais ce tableau n'entre pas dans la mémoire cache (la taille du tableau $\boldsymbol{x}$ est donnée en ligne 2 et vaut 100 000). Lorsqu'on va accéder de nouveau aux premiers éléments du tableau (lorsque on passe à une autre itération de la boucle externe ligne 1), ceux-ci auront été sortis de la mémoire cache. L'algorithme 4 réécrit l'algorithme 3 en découpant le tableau en blocs

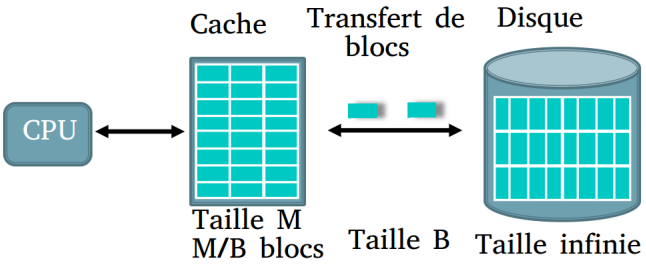


FIGURE 3.2 – Modèle *cache-aware* issu de [200].

(autrement-dit en sous-tableaux). Il est possible de choisir la taille du bloc (Algorithme 4 ligne 1) de sorte que le sous-tableau entre dans la mémoire cache. La boucle externe de l'algorithme 4 en ligne 5 parcourt les sous-tableaux ; chacun de ces sous-tableaux est réutilisé plusieurs fois (boucle en ligne 6) et pour chaque réutilisation, on ne parcourt que le sous-tableau concerné (boucle en ligne 7). L'intérêt est que si le sous-tableau entre en mémoire cache, on le charge lorsqu'on le parcourt la première fois, puis pour les neuf autres parcours, il est toujours présent en mémoire. La taille de sous-tableau optimal est dépendante de la taille de la mémoire cache [60].

Algorithme 1 : Parcours d'un vecteur sans *loop-tiling*

```
1 pour i de 1 à n faire
2     Faire une action;
```

Algorithme 2 : Parcours d'un vecteur avec *loop-tiling*

```
1 m_C = B /* Taille du bloc                          */
2 ; n_B = n/m_C /* hypothèse: n multiple de b
      */
3 pour b de 1 à n_B faire
4     pour i de 1 + b × m_C à b × m_C faire
5         Faire une action
```

Algorithme 3 : Parcours d'un tableau $\boldsymbol{x}$ plusieurs fois

```
1 pour n de 1 à 10 faire
2     pour i de 1 à 100 000
        faire
3         Accéder à x(i)
```

Algorithme 4 : Parcours d'un tableau $\boldsymbol{x}$ plusieurs fois avec *loop-tiling*

```
1 m_C = B /* Taille du bloc                          */
2 ;
3 n_B = 100 000/m_C /* hypothèse: n multiple de
      b                                              */
4 ;
5 pour b de 1 à n_B faire
6     pour n de 1 à 10 faire
7         pour i de i de 1 + b × m_C à b × m_C
            faire
8             Accéder à x(i)
```

Une des limites des algorithmes *cache-aware* est que les paramètres utilisés (comme la taille des blocs pour découper les données) pour mieux exploiter la localité des données dépendent de la taille de la mémoire cache. Or la taille de la mémoire cache varie d'un processeur à l'autre. Les performances des algorithmes *cache-aware* ne sont pas portables sur des architectures différentes. De plus, le découpage des données par bloc devient compliqué lorsqu'on considère une mémoire cache à plusieurs niveaux. C'est pourquoi plusieurs auteurs proposent plutôt de développer des algorithmes *cache-oblivious* [60].

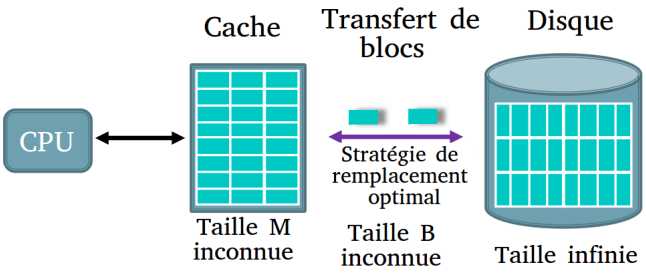


FIGURE 3.3 – Modèle *cache-oblivious* issu de [200] .

3.2.3.2 Algorithmes *cache-oblivious*

Le modèle *cache-oblivious* illustré en figure 3.3 est similaire au modèle *cache-aware* excepté que la taille de la mémoire cache et la taille des blocs transitant entre la mémoire centrale et la mémoire cache sont inconnues [70]. La taille M de la mémoire cache et la taille B des blocs transférés de la mémoire centrale vers la mémoire cache n'étant pas donné, les algorithmes *cache-oblivious* ne peuvent les utiliser pour optimiser la localité des données. En particulier, le la stratégie du *loop-tiling* ne convient pas puisque la taille des blocs est fixée en fonction de la taille de la mémoire cache. Ce modèle a deux avantages par rapport au modèle *cache-aware*. Tout d'abord, le modèle est multi-niveaux car on peut remplacer le couple mémoire centrale/mémoire cache par n'importe quel couple de niveau de mémoire : cache L_2/cache L_1, cache L_3/cache L_2 ou cache L_3/mémoire centrale. De plus, un tel algorithme est portable : puisqu'on n'utilise ni la taille du cache ni la taille des blocs (autrement dit la taille des lignes de cache), il s'adapte à n'importe quelle architecture de cache. Une autre différence est que dans ce modèle la stratégie de remplacement est supposée optimale : la ligne évincée de la mémoire cache est la ligne qui sera utilisée le plus tard possible. C'est une hypothèse de travail, irréalisable en pratique car il faudrait connaître tous les accès mémoire futurs. Une des limites du modèle *cache-oblivious* est qu'il ne prend pas en compte la prélecture (autrement dit la prédiction des accès mémoire) : un algorithme avec plus de défauts de cache mais des accès plus prédictibles peut être plus performant qu'un algorithme *cache-oblivious*.

L'alternative *cache-oblivious* de la stratégie de *loop tiling* est la stratégie de diviser pour régner qui consiste à diviser récursivement le calcul en blocs de plus en plus petits. Il existe alors un niveau de découpe pour lequel le bloc entre dans la mémoire cache, et ce sans avoir à connaître la taille de la mémoire cache. Plutôt que de diviser par bloc, une possibilité est d'utiliser une *space-filling curve* (ou courbe remplissante), qui est une courbe en dimension n remplissant l'hypercube unité de dimension n. L'idée est ensuite de découper ou de parcourir les données en suivant la *space-filling curve*. La localité obtenue en utilisant ces courbes est meilleure que la localité obtenue par un découpage récursif [200]. Par exemple, pour la multiplication de matrices denses ont été utilisées les courbe de Peano [16] et la courbe de Morton [179].

L'utilisation d'algorithmes *cache-oblivious* peut soulever plusieurs problèmes qui peuvent diminuer la performance théorique. Le premier problème est que l'accès aux données est souvent plus compliqué que dans les algorithmes *cache-aware*. Par conséquent, le processeur a du mal à prédire les accès mémoire et donc la prélecture est limitée. L'utilisation des *space-filling curve* entraîne un coût de calcul de l'adresse mémoire plus élevé, ce qui augmente alors le nombre d'instructions et donc peut cacher le gain obtenu par des accès mémoires avec une meilleure localité [200]. Le modèle *cache-oblivious* n'est pas privilégié dans cette thèse.

3.3 Ordinateurs parallèles

Un ordinateur parallèle est une machine avec plus d'un processeur qui peuvent travailler pour résoudre un problème donné.

3.3.1 Quantification du parallélisme

Il existe plusieurs mesures pour quantifier le parallélisme. La première mesure est l'accélération autrement dit le gain de temps d'une exécution parallèle par rapport à une exécution séquentielle (sur un seul processeur). Formellement, on définit l'accélération comme le ratio $S_p = \frac{T_1}{T_p}$ où T_1 est le temps d'exécution sur un seul processeur et T_p est le temps d'exécution sur p processeurs. Parfois, T_1 est pris comme le meilleur temps d'exécution séquentiel, ce qui autorise à considérer un algorithme séquentiel différent de l'algorithme parallèle. Idéalement, on souhaite une accélération linéaire c'est-à-dire que si on prend p processeurs, on veut que le temps d'exécution soit divisé par p autrement dit $T_p = \frac{T_1}{p}$. Pour mesurer à quel point l'accélération s'éloigne de cette accélération linéaire idéale, on mesure également l'efficacité comme le rapport de l'accélération sur le nombre de processeurs $E_p = \frac{S_p}{p}$ avec donc $0 < E_p \leq 1$ [60].

Il y a plusieurs raisons expliquant que l'accélération linéaire est difficile à atteindre. Tout d'abord, en utilisant plus d'un processeur, on introduit des communications entre les processeurs pour l'échange de données, communications qui ne sont pas présentes dans le cas séquentiel et qui sont une source majeure de perte d'efficacité. De plus, la charge de travail entre les processeurs peut ne pas être équitablement répartie entre les différents processeurs c'est-à-dire que les processeurs peuvent dans ce cas ne pas avoir la même charge de travail, ce qui nuit à l'accélération. Imaginons pour cela qu'on a deux processeurs et huit tâches indépendantes prenant le même temps à faire, et que pour continuer de travailler ces huit tâches doivent être terminées. Si on répartit trois tâches sur le premier processeur et cinq tâches sur le second, le premier processeur devra attendre le second sans rien avoir à faire ; en revanche, si on répartit quatre tâches chacun, les deux processeurs finiront en même temps et aucun processeur ne sera resté inactif. La dernière raison est que le code peut avoir des parties qui ne peuvent pas être parallélisées. Cela limite le parallélisme : supposons qu'il y a 5% du code qui ne peut pas être parallélisé, l'accélération est alors limitée à un facteur vingt et ce quel que soit le nombre de processeurs utilisés. Ce phénomène est connu comme la loi d'Amdhal : soit F_s la partie séquentielle du code et F_p la partie parallèle du code, $(F_s + F_p = 1)$, soit T_1 le temps d'exécution séquentiel, alors le temps T_P d'exécution sur P processeurs s'écrit $T_P = T_1(F_s + \frac{F_p}{P})$ autrement dit T_p est la somme de la partie séquentielle $(T_1 F_s)$ et de la partie qui peut être parallélisée $(\frac{T_1 F_p}{P}$. Lorsqu'on fait tendre le nombre de processeurs vers l'infini, le temps de l'exécution parallèle tend vers $T_1 F_s$. Donc l'accélération est majorée par $S_p \leq \frac{1}{F_s}$ [60].

Cependant, répartir un problème sur de plus en plus de processeurs n'a parfois pas de sens car à force de diviser le travail, il n'y a plus assez de travail pour que chaque processeur travaille efficacement. C'est pourquoi on parle parfois plus de scalabilité qui se définit comme la qualité des programmes à rester efficace pour un grand nombre de processeurs et/ou de cœurs. Cette scalabilité est pénalisée par le surcoût des communications entre processeurs, la mauvaise répartition des tâches entre les processeurs et la fraction du code parallélisé. Un programme est dit fortement scalable si son accélération est linéaire ou quasi-linéaire lorsqu'on augmente le nombre de processeurs. La scalabilité forte correspond donc à comment évolue le temps de résolution d'un problème de taille fixe en fonction du nombre de processeurs. La scabilité faible décrit comment le temps d'exécution varie en fonction du nombre de processeurs lorsque la taille des données par processeur reste fixe. Pour mesurer la scalabilité faible, lorsqu'on double le nombre de processeurs, on double aussi la taille des données [60].

3.3.2 Architectures des ordinateurs parallèles

Une façon de classer les différentes architectures existantes d'ordinateur parallèle est la classification de Flynn qui se fonde sur le partage ou l'indépendance du flot de données et du flot de contrôle. On distingue quatre types d'architecture illustrés en figure 3.4 :

— SISD (*Single Instruction Single Data*) : c'est l'architecture monoprocesseur traditionnelle : à chaque temps une seule instruction est exécutée sur un seul élément de données ;

— SIMD (*Single Instruction Multiple Data*) : chaque processeur exécute de façon synchrone la même instruction sur des données différentes et les instructions sont diffusées par une unité de contrôle ;
— MISD (*Multiple Instruction Single Data*) : très peu d'architectures répondent à cette description ;
— MIMD (*Multiple Instruction Multiple Data*) : plusieurs CPUs effectuent des instructions différentes sur des éléments de données différentes, chaque CPU ayant sa propre unité de contrôle (architecture la plus courante).

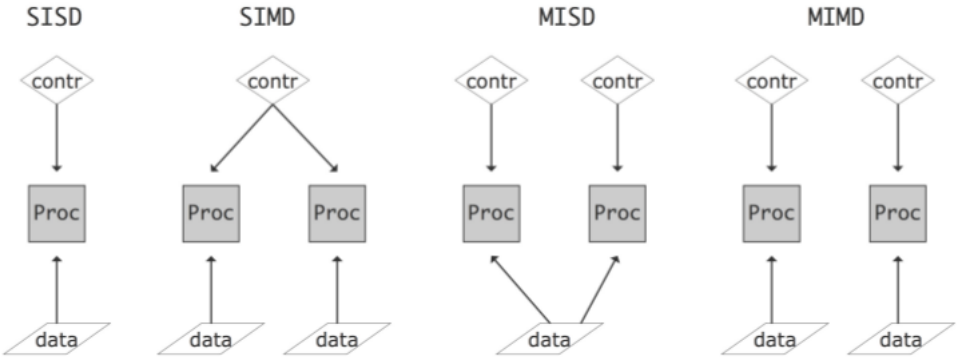


FIGURE 3.4 – Classification de Flynn issue de [60].

Les architectures SIMD sont faciles à programmer et à déboguer. Comme les processeurs sont synchronisés, le coût de synchronisation est faible. Elles sont idéales pour les opérations vectorielles. Les CPUs modernes ont des instructions vectorielles qui peuvent réaliser plusieurs instances d'une instruction simultanément. Pour les processeurs Intels, les architectures SIMD sont les *SIMD Streaming Extensions* (SSE) ou les *Advanced Vector Extensions* (AVX). Ils se trouvent également dans les *Graphical Processor Unit*s (GPUs) (**cf. section 3.3.5**) : en effet, les GPUs contiennent un grand nombre de processeurs regroupés par groupe de 32 (appelé *warp*) mais les processeurs d'un même *warp* doivent exécuter la même instruction.

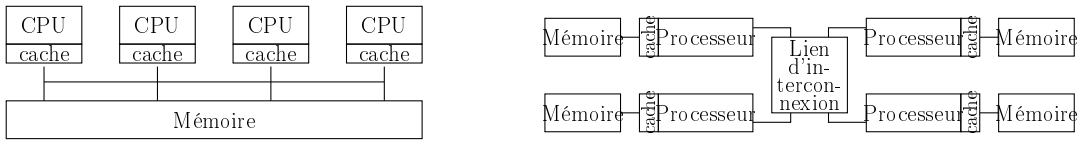
Les architectures MIMD sont plus flexibles et beaucoup plus générales. Les processeurs exécutent des instructions multiples possiblement différentes, chacun sur ses données propres. Cependant, différentes instructions ne signifient pas que les processeurs exécutent différents programmes : le mode de la majorité des architectures MIMD est en effet le *Single Program Multiple Data* où un même exécutable est lancé sur les différents processeurs. Comme les processeurs peuvent prendre des chemins différents (nombre d'itérations différent, branche différente du si), les processeurs ne sont plus synchrones à la différence de l'architecture SIMD. Il y a une grande variété d'ordinateurs avec une architecture MIMD qui diffèrent par l'organisation de leur mémoire, par le réseau connectant les différents processeurs ou les façons de les programmer. Les clusters de calcul réunissant plusieurs processeurs suivent l'architecture MIMD puisque les processeurs sont indépendants.

Un ordinateur parallèle a été défini comme un ensemble de processeurs travaillant ensemble pour résoudre un même problème. Cela suppose que les processeurs ont accès aux mêmes données. On distingue donc également les architectures en fonction de l'organisation de leur mémoire. Il existe quatre types d'architecture de mémoire [84] :

— les architectures à mémoire partagée,
— les architectures à mémoire distribuée,
— les architectures mixtes
— les architectures hybrides (GPU).

3.3.3 Architecture à mémoire partagée

Sur une architecture à mémoire partagée, tous les processeurs partagent la même mémoire : ils partagent le même espace d'adressage. Si deux processeurs font référence à une variable x, ils accèdent au même emplacement mémoire. Les processeurs ont en revanche leur propre mémoire cache dans laquelle est copiée une partie de la mémoire globale et dont il faut assurer la cohérence entre les processeurs. On distingue deux types d'architecture à mémoire partagée : l'architecture UMA (*Unified Memory Access*) et l'architecture NUMA (*Non-Unified Memory Access*) (illustrées en figure 3.5). Pour l'architecture UMA, il y a une seule mémoire centralisée et le temps d'accès à un emplacement quelconque de la mémoire est identique quel que soit le processeur qui accède à la donnée. Cependant, l'architecture UMA est limitée à un petit nombre de processeurs. Dans l'architecture NUMA, il y a une mémoire centrale par processeur (ou par groupe de processeurs UMA) mais les processeurs peuvent accéder à n'importe quel emplacement mémoire. Cependant, le temps d'accès à la mémoire dépend d'où se trouve la donnée en mémoire : un processeur accède plus rapidement aux données présentes dans sa mémoire centrale qu'aux données stockées dans la mémoire centrale d'un autre processeur. La mémoire est physiquement distribuée mais elle est logiquement partagée, c'est-à-dire que l'espace d'adressage est partagé entre les processeurs. La gestion de la cohérence de cache est le plus souvent transparent pour l'utilisateur. L'échange de données entre les tâches attribuées aux processeurs est transparent et rapide. Mais le nombre de processeurs par machine est limité et les performances décroissent rapidement avec le nombre de processeurs à cause du trafic généré par les accès mémoire. De plus, lorsque le nombre de processeurs augmente, les machines deviennent coûteuses.



(a) Architecture UMA (b) Architecture NUMA

FIGURE 3.5 – Architecture à mémoire partagée .

3.3.4 Architecture à mémoire distribuée

Pour les architectures à mémoire distribuée, un espace mémoire est associé à chaque processeur et les processeurs sont connectés entre eux à travers un réseau comme illustré en figure 3.6. L'accès à la mémoire du processeur voisin doit donc se faire explicitement par échange de message à travers le réseau d'interconnexion entre les processeurs. Le réseau d'interconnexion est important car il détermine la vitesse d'accès aux données d'un processeur voisin. Il est caractérisé par sa latence qui correspond au temps d'initialisation d'une communication, sa bande passante qui correspond à la vitesse de transfert des données à travers le réseau et sa topologie qui est l'architecture physique du réseau. L'accès à la mémoire locale est rapide. L'architecture permet d'obtenir facilement des machines avec un grand nombre de cœurs pour un coût réduit par rapport à l'architecture à mémoire partagée. En revanche, l'échange de données doit être géré explicitement par le programmeur et les performances sont dépendantes de la qualité du réseau entre les processeurs.

3.3.5 Graphical Processor Unit (GPU)

Les GPUs ont été développés originellement pour leurs performances graphiques pour les jeux vidéo ou les logiciels de conception assistée par ordinateur. Le problème d'origine était l'affichage d'un maillage 3D texturé à 60-100 frames par secondes lorsque le maillage compte plus de dix millions de triangles ou lorsque la résolution de l'écran est portée à plus de huit millions de

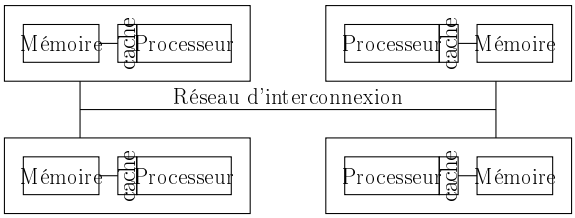


FIGURE 3.6 – Architecture à mémoire distribuée .

pixels. En 1999, NVIDIA introduit des cartes graphiques 3D appelées GPUs non programmables qui permettent d'augmenter la résolution de l'écran et le nombre de triangles dans le maillage. Avec la carte graphique NVIDIA GeForce 3, est introduit un pipeline programmable : c'est le début des GPGPUs (*General Purpose Computing on Graphical Processing Units*) et du calcul scientifique sur GPU. Mais la programmation reste complexe car il faut transformer les données en un format graphique (pixel ou sommet des triangles). Fin 2006, NVIDIA met sur le marché la GeForce 8800 avec une API dédiée au calcul scientifique à côté de l'API graphique et permet l'utilisation du langage C pour la programmation. Il existe aujourd'hui deux vendeurs de cartes graphiques : NVIDIA et AMD. Les GPUs NVIDIA sont programmables en CUDA (Compute Unified Device Architecture) langage créé par NVIDIA pour ses GPUs . Les GPUs de AMD se programment en OpenCL (Open Computing Language) qui est un langage de programmation dérivé du C, proposé comme un standard ouvert et conçu pour programmer des systèmes parallèles hétérogènes comprenant par exemple à la fois un CPU multi-cœur et un GPU. Par la suite, nous nous intéressons spécifiquement aux GPUs proposés par NVIDIA [227].

La conception générale d'un GPU est donc motivée par le pipeline graphique : des opérations identiques sont réalisées sur un grand nombre d'éléments sous forme d'un parallélisme de données et plusieurs blocs d'un tel parallélisme de données sont actifs simultanément. Cependant comme le CPU, le GPU est limité par les accès mémoires qui entraînent une longue latence. Mais là où le CPU réduit la latence en insérant plusieurs mémoires cache, le GPU choisit de délivrer de grandes quantités de données en même temps avec un taux moyen élevé plutôt que de délivrer un seul résultat le plus vite possible. Cela est possible en supportant un grand nombre de threads et en pouvant changer de contexte entre les threads très rapidement. Le GPU cache la latence des accès mémoire par l'exécution d'autres threads : quand un thread attend de recevoir une donnée depuis la mémoire, un autre thread pour lesquels les données sont disponibles effectue son calcul [60]. La figure 3.7 montre les performances théoriques en GFlops des GPUs et des CPUs : les performances des GPUs sont bien meilleures que les CPUs. En effet, le GPU a une structure hautement parallèle : il dispose de plusieurs centaines de cœurs et de plusieurs milliers de threads. Les GPUs sont particulièrement adaptés pour les problèmes où le même programme (avec plus d'opérations arithmétiques que d'opérations mémoire) s'exécute en parallèle sur plusieurs éléments des données. D'une part, puisque le même programme est exécuté sur chaque élément, le contrôle de flux est moins sophistiqué. D'autre part, comme le programme est effectué sur un grand nombre d'éléments et qu'il y a plus d'opérations arithmétiques que d'opérations mémoire, la latence peut être cachée par des calculs plutôt qu'avec la présence de mémoire cache : dès qu'un thread est en attente d'un accès mémoire, il est remplacé par un autre thread qui peut exécuter une opération. La figure 3.8 montre l'architecture schématique d'un CPU et d'un GPU. Il existe plusieurs différences entre le CPU et le GPU. Premièrement, le GPU est attaché au CPU, relié au CPU par un bus PCI-X donc pour opérer sur des données, il faut que le CPU ait copié ces données dans la mémoire GPU. Or la bande-passante entre le CPU et le GPU est faible, plus faible même que le transfert des données à l'intérieur du GPU, il faut donc disposer de suffisamment de calculs à faire sur GPU pour compenser le temps de transfert du CPU vers le GPU. De plus, les GPUs ont mis l'accent sur le calcul en simple-précision ; et donc bien que le calcul en double précision ait été introduit sur les GPUs, la vitesse de calcul en double précision

est deux fois moindre qu'en simple précision. Le CPU est optimisé pour gérer un seul flot d'instructions potentiellement hétérogènes ; *a contrario*, un GPU a été créé pour le parallélisme des données. Une autre différence est que le CPU est prévu pour gérer un thread ou un petit nombre de threads alors que le GPU a besoin d'un grand nombre de threads (supérieur au nombre de cœurs disponibles sur GPU) pour avoir une bonne performance.

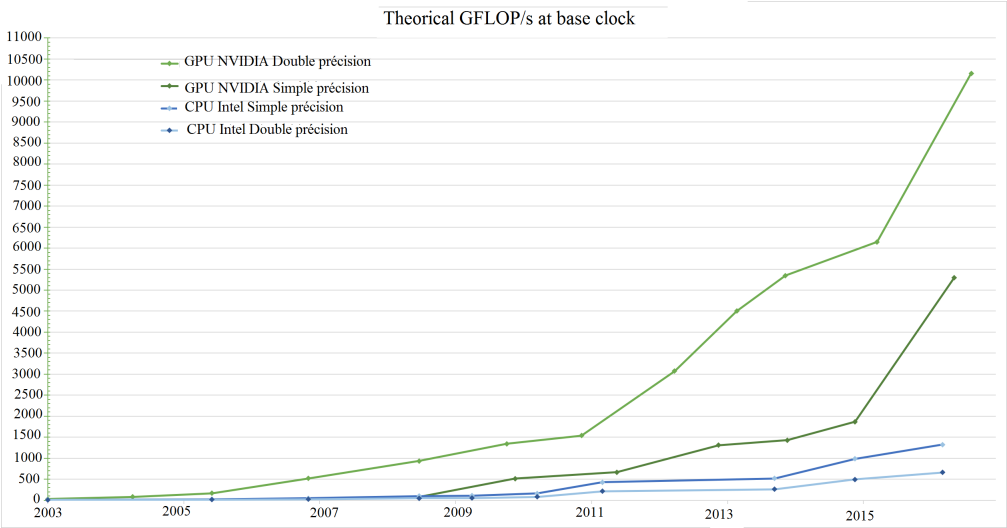


FIGURE 3.7 – Comparaison des performances en GFlops des GPUs et des CPUs issue de [51].

FIGURE 3.8 – Architecture shématique des CPUs et des GPUss issue de [51]
ALU : unité en charge de fonctions basiques de calcul arithmétique et les opérations logique.

Un GPU est composé d'un grand nombre de cœurs pour le calcul avec plus de cœurs en simple précision qu'en double précision. Ces cœurs de calcul sont regroupés en *Streaming Multiprocessors* (*Streaming Multiprocessor* (SM)) qui opèrent de manière indépendantes les uns des autres. Un SM est donc constitué de plusieurs cœurs qui partagent un même flux d'instructions : les SMs agissent comme des processeurs vectoriels. Chaque cœur a des registres privés. Les cœurs dans un même SM partagent une mémoire L1 rapide. L'ensemble des SMs ont accès à la mémoire globale et à des mémoires spécialisées : la mémoire de textures et la mémoire constante [104].

La figure 3.9 présente le modèle mémoire des GPUs NVIDIA. On renvoie à la section 3.4.3 pour une définition des blocs. La mémoire globale est la mémoire principale du GPU. Elle est accessible via des pointeurs en arguments des noyaux. L'allocation en mémoire globale se fait uniquement depuis l'hôte. Lorsqu'elle est accédée depuis un noyau, les accès doivent être coalescents c'est-à-dire que les accès qui se font par segment de 32, 64 ou 128 octets doivent se suivre en mémoire [67]. La mémoire partagée est accessible par tous les threads d'un même bloc (**cf. section 3.4.3** pour une définition d'un bloc), de manière plus rapide que la mémoire globale. Son allocation est statique et elle ne peut être accédée que depuis un noyau CUDA. Elle est utilisée comme un cache pour partager les données entre les threads. La mémoire de texture est physiquement dans la mémoire globale. Cependant, elle diffère de la mémoire globale : elle est

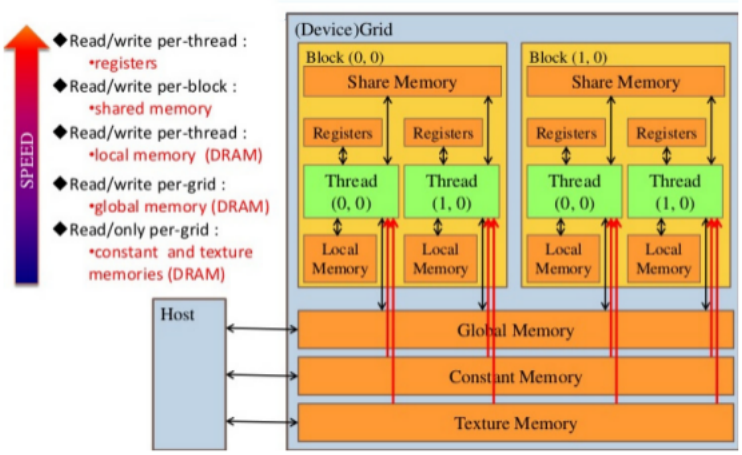


FIGURE 3.9 – Organisation mémoire du GPU issue de [67].

optimisée pour la localité 2D et peut s'utiliser en 1D, 2D ou 3D. Contrairement à la mémoire globale, l'accès à la mémoire texture ne se fait pas par des pointeurs mais par des fonctions spécifiques. La mémoire locale est propre à chaque thread utilisé par le compilateur mais elle n'est pas programmable. Elle sert à stocker les tableaux locaux avec des accès non constants ou des registres dépassant la capacité mémoire par exemple. La mémoire constante est située dans la mémoire globale (partagée entre tous les blocs) mais elle n'est accessible qu'en lecture seule depuis un noyau et en contrepartie, l'accès est très rapide pour des paramètres déclarés comme constants dans le noyau.

3.3.6 Autres architectures

L'architecture mixte est la combinaison des architectures à mémoire partagée et à mémoire distribuée. Les machines sont alors constituées de machines à mémoire partagée (nœud de calcul) reliées entre elles par un réseau d'interconnexion. Les architectures hybrides correspondent à l'équipement des nœuds de calcul avec des GPUs.

3.4 Modèle de programmation

À chaque architecture de machine parallèle correspond un modèle de programmation. Nous détaillons ici le modèle de programmation pour les architectures à mémoire partagée et pour les architectures à mémoire distribuée. Les architectures mixtes combinent les modèles de programmation des architectures à mémoire partagée et à mémoire distribuée. Le modèle de programmation pour les GPUs est présenté en section 3.3.5.

3.4.1 Programmation à mémoire partagée

La programmation pour des architectures à mémoire partagée s'effectue sur le modèle du multithreading. Un programme multithreading s'exécute dans un processus unique et active plusieurs processus légers (**cf. section 3.2.1**) capables de s'exécuter de manière concurrente. Les processus légers sont dynamiques c'est-à-dire qu'ils peuvent être créés au cours de l'exécution du programme. Lorsqu'un programme commence, il y a un seul processus léger appelé processus maître. Les autres processus légers sont lancés de façon dynamique et le processus maître attend qu'ils aient fini leur exécution avant de poursuivre le programme. C'est le modèle *fork-join* montré en figure 3.10. Sur la figure, la ligne rouge correspond au processus maître. Il y a ici deux régions exécutées en parallèle. Pour la première région, on crée trois processus légers (traits

bleus) et l'un d'eux crée pendant son exécution deux autres processus légers (lignes vertes).À la fin de sous-section parallèle, les processus légers représentés en vert sont détruits. Puis les trois processus légers (en bleus) sont détruits à la fin de la première section parallèle. Le processus maître continue l'exécution du programme en séquentiel. Puis il rencontre une seconde section parallèle et cette fois-ci sept processus légers sont créés (en bleu). Une fois la section terminée, ils sont détruits et le processus maître reprend l'exécution du programme en séquentiel. Les processus légers ont accès à la mémoire du processus qui les lance et également à une mémoire locale qui leur est propre et qui est invisible pour les autres processus légers. Les processus légers constituent un moyen de faire du parallélisme. Cependant, pour que l'utilisation des processus légers soit plus rapide qu'une exécution sur un seul processus léger, il faut que le matériel puisse lancer plus d'un processus léger à la fois. Cela est possible lorsqu'un processeur multicœur peut lancer un processus léger par cœur ou lorsqu'il est possible de lancer plusieurs processeurs légers par cœur. C'est donc l'exécution des processus légers sur plusieurs processeurs ou cœurs qui permet l'exécution en parallèle du programme. Le système distribue les processus légers sur les différents processeurs ou cœurs de la machine à mémoire partagée [60, 84].

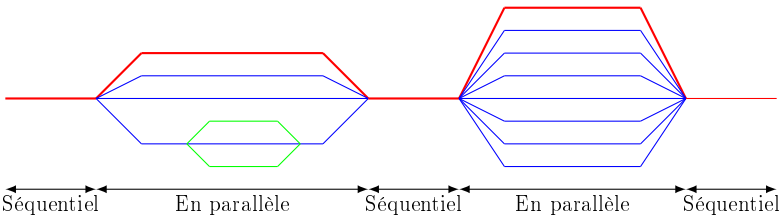


FIGURE 3.10 – Création et suppression des processus légers au cours d'une exécution parallèle .

On appelle contexte la totalité des données auxquelles peut accéder un processus léger. Le contexte contient donc à la fois les données partagées et les données privées propres au processus léger, ainsi que les résultats temporaires des calculs. Si on crée plus de processus légers que le nombre de cœurs qu'a le processeur, le processeur peut avoir besoin de passer d'une exécution d'un processus léger à l'exécution d'un autre processus léger. Un tel changement est appelé la commutation de contexte et est en général coûteux sur la plupart des processeurs. Comme les processus légers accèdent aux mêmes données, il est possible que des processus légers écrivent dans la même variable. Par exemple, le produit scalaire entre deux vecteurs $\boldsymbol{a}$ et $\boldsymbol{b}$ se calcule en faisant $s = s + \boldsymbol{a}(i) \times \boldsymbol{b}(i)$. Les produits $\boldsymbol{a}(i) \times \boldsymbol{b}(i)$ sont indépendants et peuvent donc s'exécuter en parallèle ; mais les processus légers doivent tous écrire dans la variable s. Pour que le code en parallèle se comporte de la même manière en parallèle qu'en séquentiel, il faut donc faire attention aux écritures dans la mémoire partagée. Pour cela, il existe deux dispositifs : les sections critiques ou les verrous. Une section critique est une partie du code qui ne peut être exécutée que par un processus léger à la fois : il faut que l'exécution de la section critique soit totalement terminée avant qu'un autre processus léger puisse entrer dans la section critique. La deuxième solution est de poser un verrou sur une partie des données. Lorsqu'un processus léger veut accéder à une donnée protégée par un verrou, deux cas de figures sont possibles : soit aucun autre processus léger n'a le verrou des données et alors le processus léger prend le verrou et peut travailler sur les données, soit un autre processus léger a déjà pris le verrou et le processus léger doit attendre que le verrou soit libéré [60].

Nous avons vu en section 3.3.3 que dans le cas d'une architecture NUMA, l'accès aux données n'est pas uniforme et dépend d'où est localisée la donnée. Pour améliorer les performances d'un code parallèle, il est préférable qu'un processus léger travaille sur les données de la mémoire centrale du cœur auquel il est attaché. De même, supposons qu'il y a deux sections parallèles, la première créant la donnée et la seconde exploitant la donnée, pour une donnée en particulier, il faut s'assurer que le processus léger créant la donnée et le processus léger travaillant dessus soient sur le même cœur. On appelle affinité la mise en correspondance entre les processus légers

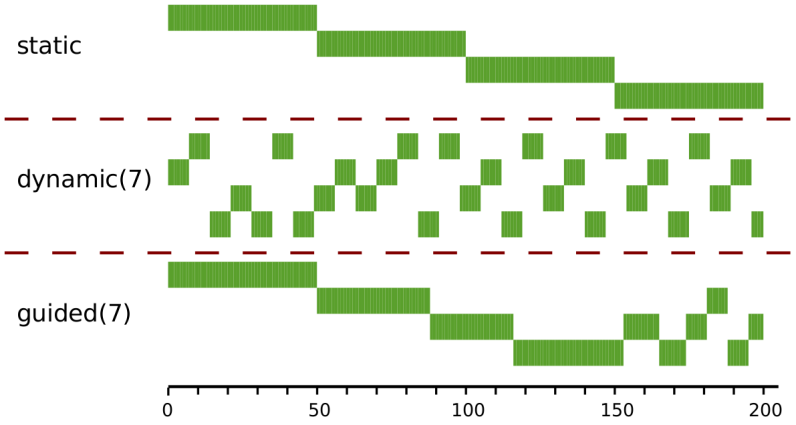


FIGURE 3.11 – Différentes répartitions des itérations d'une boucle en OpenMP issue de [30]. Chaque ligne représente un thread .

et les cœurs.

OpenMP (Open Multi-Processing) est une interface de programmation pour générer un programme multithreads. Ce n'est pas un langage de programmation mais une extension des langages de programmation C et Fortran. Sa principale approche du parallélisme est l'exécution en parallèle des boucles. OpenMP est basé sur les processus légers et offre donc un parallélisme dynamique c'est-à-dire que le nombre de processus légers s'exécutant en parallèle peut varier d'une portion de code à l'autre. Le parallélisme est déclaré en créant des régions parallèles. Il fonctionne ensuite par l'insertion de directives dans le code source, directives qui sont ensuite interprétées par le compilateur. Le système d'exécution gère ensuite l'exécution en parallèle. Les directives se mettent sous la forme de commentaires spécifiques commençant en Fortran par !$OMP et en C par #pragma omp. De cette façon, un code OpenMP ressemble à un code Fortran ou C si le compilateur ne supporte pas OpenMP. Du coup, pour que le programme s'exécute en parallèle, il doit être lié à la librairie d'exécution OpenMP. Il est ainsi beaucoup plus facile d'écrire un code en OpenMP qu'en MPI car on peut partir d'un code séquentiel et y ajouter des directives pour le rendre parallèle [30]. OpenMP propose plusieurs manières de répartir les itérations d'une boucle parallèle :

— de manière statique : les itérations sont découpées en blocs d'une taille spécifiée par l'utilisateur ou de taille égale au ratio du nombre d'itérations sur le nombre de processus légers et les blocs sont répartis de manière cyclique entre les processus légers ;
— de manière dynamique : les itérations sont découpées en blocs d'une taille spécifiée par l'utilisateur ou de taille égale au ratio du nombre d'itérations sur le nombre de processus légers ; mais les blocs sont répartis de manière dynamique entre les processus légers : dès qu'un processus léger a fini le bloc qui lui a été alloué, un nouveau bloc lui est dynamiquement alloué ;
— de manière guidée : pour une taille de bloc égale à 1, la taille de chaque bloc est proportionnelle aux nombres d'itérations qui n'ont pas encore été assignées à un processus léger) divisé par le nombre de processus légers, puis on enlève un à la valeur du ratio. Pour une taille de bloc égale à k, la taille des blocs est déterminée de la même façon mais on ajoute la contrainte que les blocs ne peuvent pas contenir moins de k itérations. La taille des blocs varie donc au cours de l'exécution.

La répartition statique n'est pas toujours optimale car les itérations peuvent avoir un temps d'exécution différent. La figure 3.11 montre les trois types de répartition pour une boucle avec 200 itérations.

3.4.2 Modèle de programmation à mémoire distribuée

Pour les architectures à mémoire distribuée, on utilise le modèle de programmation par message. En effet chaque processus exécute un programme sur des données différentes et chacun n'a accès qu'aux données dans sa mémoire centrale sans accès direct aux données présentes dans les mémoires des autres. Cependant, ils peuvent s'échanger des messages soit pour se transférer des données soit pour se synchroniser. La programmation par message nécessite des outils de sécurité et de gestion des droits d'accès, de créations de processus à distance, de communication entre processus et de synchronisation des processus. On peut également trouver des outils pour la gestion de la cohérences des données et des traitements, un séquenceur des tâches réparties et des outils de gestion dynamique des processeurs et des processus (gestion des pannes et des points de reprise). Dans un modèle de programmation par message, le parallélisme et la distribution des données est à la charge du programmeur. De plus, le programmeur doit gérer de manière explicite l'échange de données entre les processeurs. MPI (Message Passing Interface) est une interface de programmation pour générer un programme par échange de messages, qui est devenu le standard de ce mode de programmation. L'interface fournit des fonctions permettant de gérer un environnement d'exécution (création de l'environnement parallèle, création de groupe de processeurs), de communiquer point-à-point autrement dit entre deux processus : ce sont essentiellement des variantes des fonctions d'envoi et de réception de messages et des fonctions pour les communications collectives. On distingue plusieurs types de communication collective : *one-to-many* pour les communications d'un processus vers un ensemble de processus ; *many-to-one* : un processus collecte une information en provenance d'un ensemble de processus et *many-to-many* qui est un échange global d'informations entre plusieurs processus [8]. Un communicateur est un ensemble statique de processus, qui peut être créé et détruit en cours d'application et qui est utile pour les communications collectives. Chaque processus est identifié par un numéro d'instance (rang dans un groupe) et par le communicateur. Un processus peut appartenir à plusieurs communicateurs et avoir un rang différent dans chacun d'eux [151].

Un message est divisé en une zone de données et une enveloppe. Les données contiennent l'adresse de la mémoire tampon (buffer), le nombre d'éléments et le type des données. L'enveloppe d'un message doit permettre la caractérisation et le traitement du message. Elle contient donc le communicateur concerné, le numéro de l'émetteur, le numéro du récepteur, l'étiquette du message et la taille du message. Il existe plusieurs modes de communication pour l'envoi et la réception de messages. Les envois et réceptions peuvent être synchrones c'est-à-dire que le premier arrivé attend le second. Ils peuvent être *a contrario* asynchrones : l'émetteur et le récepteur ne s'attendent pas. Un envoi asynchrone peut toutefois être bloqué par la non-consommation du message par le récepteur. L'émetteur et le récepteur n'ont pas à être tous les deux synchrones ou tous les deux asynchrones. La figure 3.12 montre l'implémentation d'un algorithme de factorisation LU de matrices creuses (avec un grand nombre de zéros) asynchrone (développé par l'ENSEEIHT,Toulouse) et synchrone (développé au NRSC à Berkeley) : les phases de calcul sont colorées en bleu et les blocs rouges correspondent aux communications. L'algorithme asynchrone arrive à recouvrir une partie des communications par des calculs. L'envoi et la réception peuvent également être bloquants : la ressource est disponible en retour de la procédure autrement dit, la donnée est modifiable dès que le processus reprend la main sans que le message envoyé soit modifié. La réception peut être non-bloquante simple : un paramètre de retour indique si l'information est disponible. Les envois et réceptions peuvent aussi être non-bloquants généraux : la procédure rend la main au processus immédiatement sans garantie que la donnée ait été envoyée ou reçue. Dans ce cas, l'utilisateur ne peut pas réutiliser l'espace mémoire associé au message sans risque de changer ce qui sera envoyé. Il faut donc tester et attendre la libération (dans le cas d'un envoi) ou la réception effective de la donnée. La figure 3.13.(a) montre l'envoi asynchrone dans les cas bloquant et non bloquants. Dans le cas d'un envoi bloquant, si le message est de petite taille, l'émetteur reprend immédiatement la main et l'envoi devient non-bloquant. Si l'envoi est synchrone, l'émetteur doit attendre que le récepteur appelle une fonction de ré-

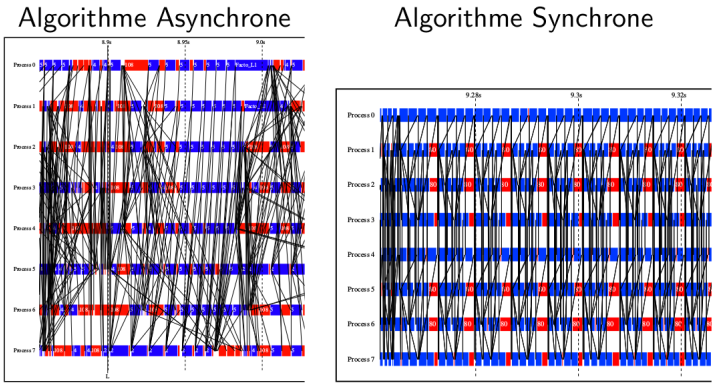
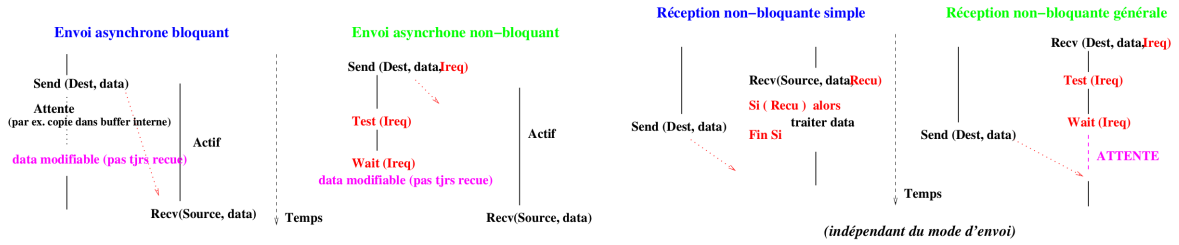


FIGURE 3.12 – Algorithme asynchrone vs Algorithme synchrone, image issue de [8].



(a) Envoi asynchrone (b) Réception

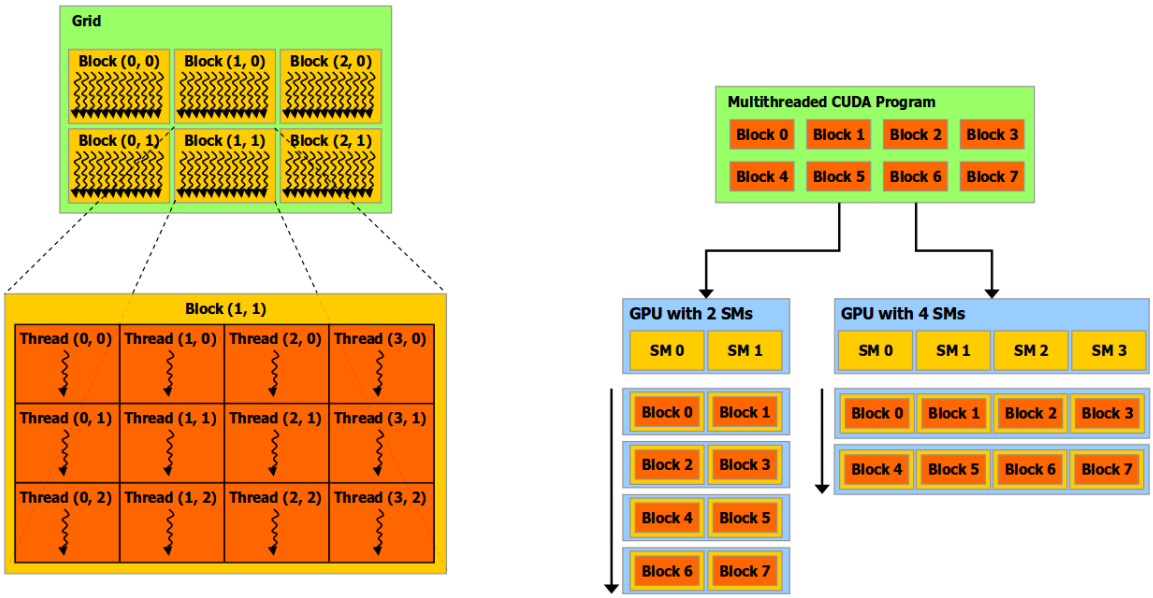
FIGURE 3.13 – Mode bloquant et non-bloquant issu de [8].

ception. Les données sont alors copiées directement dans la zone de réception du destinataire. La figure 3.13.(b) montre la réception en mode non-bloquant simple et en mode non-bloquante générale. La communication entre les processeurs est beaucoup plus lente que le transfert des données depuis la mémoire d'un seul processeur. Pour cette raison, il faut s'assurer qu'il y a assez de travail pour chaque processeur pour compenser les coûts de communications ou il faut recouvrir les temps de communication avec des calculs pour cacher la latence.

3.4.3 CUDA pour les GPUs

Le programme principal s'exécute sur le CPU. Une partie du code va s'exécuter sur le CPU (en séquentiel ou en parallèle) : des calculs qu'on souhaite exécuter sur le CPU car il est difficile de le paralléliser sur GPU ou car il n'y a pas assez de travail pour justifier l'usage du GPU, la gestion de la mémoire sur CPU et sur GPU, des synchronisations voire des communications. Mais le programme principal peut également lancer des noyaux (*kernel*) sur le GPU : les noyaux sont des codes écrit spécialement pour s'exécuter sur le GPU. Une séquence classique de code CUDA est : le CPU alloue de la mémoire sur le GPU, puis il copie les données dans la mémoire globale du GPU allouée, le CPU lance ensuite l'exécution du noyau sur GPU, le GPU exécute le noyau puis le CPU copie les résultats depuis la mémoire globale du GPU. Les noyaux GPU sont exécutés par plusieurs threads en parallèle : tous les threads exécutent le même code : ils effectuent la même opération mais sur des données différentes. Chaque thread est identifié de manière unique. Le modèle CUDA regroupe les threads en blocs de threads, typiquement une centaine de threads par bloc. À l'intérieur d'un bloc de threads, les threads peuvent interagir : ils peuvent s'échanger des données via la mémoire partagée et ils peuvent être synchronisés. Les blocs de threads sont indépendants et doivent donc pouvoir être exécutés dans n'importe quel ordre. Ces blocs de threads sont eux-mêmes groupés en une grille comme montré en figure 3.14.(a). Le bloc de threads est identifié de manière unique dans une grille. La grille est un tableau de

blocs exécutant le même noyau et peut accéder à la mémoire globale du GPU. Les blocs ne se synchronisent qu'en terminant un noyau et en commençant un nouveau noyau. Un noyau est exécuté sur une grille pour laquelle il est possible de spécifier le nombre de blocs par grille et le nombre de threads par bloc. Chaque thread est exécuté par un cœur du GPU. Un bloc de threads est exécuté sur un seul SM : il peut y avoir plus de threads dans un bloc que de cœurs disponibles sur le SM, ce qui assure que le GPU a suffisamment de travail. Il est possible d'avoir plusieurs blocs de threads s'exécutant sur un même SM, l'avantage est qu'ainsi, les blocs de threads vont être échangés au cours de l'exécution pour cacher la latence. L'indépendance des blocs de threads assure la scalabilité du code CUDA : en effet si on exécute huit blocs et qu'on dispose de deux SMs, chaque SM est chargé de quatre blocs, si on dispose de quatre SMs, chaque SM est alors chargé de deux blocs, ce qui est illustré en figure 3.14.(b) [51, 104, 227].



(a) Thread,Bloc et Grille

(b) Scalabilité du GPU

FIGURE 3.14 – Organisation logique du GPU : threads, blocs et grille issu de [51].

Cependant, les threads ne sont pas créés égaux. Les SMs sont de vrais processeurs vectoriels de largeur trente-deux, c'est-à-dire que trente-deux cœurs agissent en mode bloqué plutôt qu'indépendamment et partagent donc le même flot d'instructions avec un seul compteur de programme. Les threads à l'intérieur d'un bloc sont organisés en groupe de trente-deux appelé *warp*. Les threads de chaque *warp* exécutent une seule et même instruction. Si un bloc contient plusieurs *warps*, les *warps* sont exécutés tour à tour. Les opérations mémoire sont également faites sur la base d'un *warp*, autrement dit par bloc de trente-deux adresses mémoires consécutives à la fois. Les branchements (conditionnel par exemple) peuvent donc être pénalisants pour le parallélisation car les trente-deux threads d'un même *warp* exécutent la même instruction. Si au sein d'un *warp*, certains threads exécutent à cause d'un branchement une instruction différente des autres threads du *warp* : chaque instruction différente est traitée séquentiellement (et non plus simultanément). Pour cela, les threads qui n'ont pas pris le chemin ne cours sont désactivés. Dans le pire des cas, chaque thread d'un *warp* peut suivre un chemin qui lui est propre ce qui conduit à l'exécution de trente-deux chemins en séquentiel. En revanche, les threads d'un *warp* et les threads d'un autre *warp* peuvent exécuter des chemins différents en parallèle [104].

Une application CUDA peut exécuter différentes tâches simultanément : des calculs sur l'hôte, des calculs sur le GPU, des transferts mémoires CPU vers GPU et des transferts mémoires GPU vers CPU. Les possibilités dépendent d'une caractéristique du GPU : la *Compute Capability*

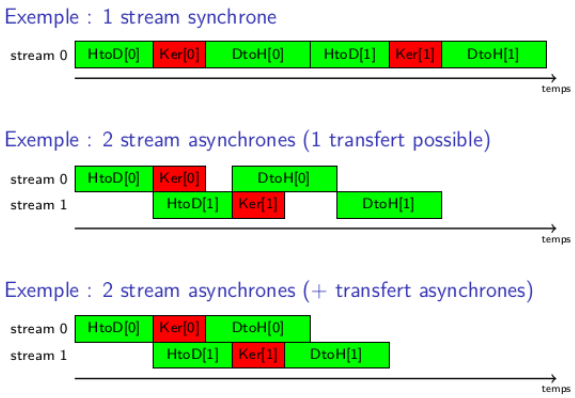


FIGURE 3.15 – Exemple d'utilisation de *streams* issu de [211].

(CC). Certaines opérations sont de plus considérées comme asynchrones (il n'y a pas d'attente entre les deux exécutants) comme le lancement des noyaux (le CPU récupère donc la main une fois le noyau GPU lancé) et les copies mémoires déclarées asynchrones. Un GPU (CC>2.x) peut exécuter différents noyaux simultanément. On peut également superposer l'exécution de noyau et un transfert mémoire si on utilise la mémoire *page-locked* sur l'hôte. Un GPU (CC>2.x) peut exécuter des copies CPU vers GPU et GPU vers CPU simultanément, si la mémoire allouée sur CPU est *page-locked*. Un *stream* est une série de différentes commandes (noyaux et copies mémoire) qui s'exécutent dans l'ordre d'appel sur le GPU. Les commandes de différents *streams* peuvent s'exécuter dans un ordre quelconque ou simultanément sans garantie. La figure 3.15 montre comment l'utilisation de plusieurs *streams* asynchrones réduit le temps d'exécution du programme CUDA en permettant à la copie et au noyau de s'exécuter simultanément. Cependant, si on ne précise pas le *stream* sur lequel s'exécute un noyau, le noyau est lancé sur le *stream* par défaut qui a effet de synchronisation implicite. En effet, toutes les opérations effectuées sur le *stream* par défaut sont synchrones. L'asynchronisme peut donc être atteint en utilisant plusieurs *streams* (différents du *stream* par défaut), en copiant les données de manière asynchrone avec la mémoire hôte *page-locked* et si les ressources sont disponibles. Certaines opérations sont responsables d'une synchronisation implicite : l'utilisation du *stream* par défaut, les allocations mémoires paged-locked sur CPU, les allocations dans la mémoire globale du GPU et les copies synchrones. Il existe ensuite plusieurs synchronisations explicites. On peut synchroniser toutes les opérations s'exécutant sur GPU : on attend la fin de l'ensemble des opérations lancées sur le GPU. On peut également synchroniser un seul *stream* : on attend la fin de toutes les opérations sur un *stream*. Enfin, un *stream* peut être synchronisé à l'aide d'un événement : le *stream* attend que l'événement spécifié arrive avant de continuer son exécution. Les événements permettent de mesurer précisément la progression d'une application. Un événement est ajouté comme une opération dans un *stream* et obtient la date de sa complétion lorsque toutes les opérations qui le précèdent sont terminées. [211].

Chapitre 4

Objectifs de la thèse

4.1 Objectifs

Cette thèse a pour objectif de proposer une méthode d'analyse guidée par les données applicable à des acquisitions en IRMf alimentaire en tenant compte de la spécificité d'un tel paradigme. Pour cela nous disposons d'un jeu de données de référence acquis pour la thèse qui permet d'évaluer les perturbations induites par les stimulations gustatives répétées. Ce jeu de données nous a permis d'analyser les perturbations et de proposer une méthode pour corriger ces perturbations. En s'appuyant sur la littérature, nous avons choisi de parcelliser les données à l'aide d'un clustering hiérarchique agglomératif avec le critère de Ward. En particulier un des enjeux de la thèse est de permettre la parcellisation d'un sujet sur l'ensemble du cerveau (même pour des images à haute résolution spatiale) en quelques heures en considérant l'ensemble de la réponse temporelle. Le développement d'une méthode efficace de parcellisation passe par le développement d'algorithmes parallèles. Enfin, cette thèse a pour objectif de mettre en œuvre une analyse multi-échelle des données, c'est-à-dire de regarder le résultat de la parcellisation pour différents nombres de parcelles. L'objectif de cette analyse multi-échelle est d'identifier des régions codant les caractéristiques du stimulus alimentaire utilisé ; et de pouvoir raffiner la localisation de ces régions en des sous-régions plus fines.

4.2 États de l'art

Sont regroupés ici trois états de l'art. Nous établissons d'abord un rapport sur les différentes techniques de correction du bruit contaminant les données d'IRMf présenté en section 1.2.3. Nous présentons ensuite les travaux sur la parallélisation des algorithmes de clustering hiérarchique agglomératif (sans se restreindre au critère de Ward) et les travaux sur la parallélisation du calcul de la distance euclidienne, qui représente une part importante de l'algorithme de Ward.

4.2.1 Correction du bruit

Nous présentons ici les différentes méthodes pour réduire le bruit physiologique et pour tenir compte des mouvements de la tête. Les mouvements de la tête posent particulièrement problème lorsqu'ils sont corrélés avec la tâche expérimentale (discours manifeste ou déglutition par exemple [25]) car ils peuvent empêcher la différentiation entre le signal BOLD et le mouvement malgré le moyennage du signal sur les différentes répétitions de la tâche. Pelphrey *et al.* soulignent qu'une corrélation entre un artéfact de mouvement et le stimulus est susceptible de biaiser gravement les résultats [57].

4.2.1.1 Bruits physiologiques

Les bruits physiologiques peuvent être *aliasés* et donc mélangés au signal d'intérêt en fonction du temps de répétition choisi. L'inconvénient du phénomène d'*aliasing* est que les bruits physiologiques ne peuvent pas être enlevés par filtrage temporel. On distingue trois approches pour tenir compte du bruit physiologique. La première approche consiste à utiliser un enregistrement externe de la pulsation cardiaque (à l'aide d'un oxymètre par exemple) et de la respiration (à l'aide d'une ceinture pneumatique) pour réduire le bruit physiologique grâce à un filtre coupe-bande ou grâce au logiciel RETROICOR [77]. Lorsqu'aucun enregistrement n'est disponible (car non acquis ou inexploitable), le bruit physiologique peut être estimé en moyennant le signal dans des régions dominées par le bruit physiologique comme les larges vaisseaux sanguins, les ventricules ou la substance blanche. Enfin, des approches conduites par les données ont aussi été développées pour enlever le bruit physiologique. Par exemple, le logiciel CompCor utilise l'analyse en composantes principales (voir section 2.2.1 pour une explication de la méthode) pour éliminer le bruit physiologique [21]. Les algorithmes CORSICA [165] et PESTICA [18] emploient l'analyse en composantes indépendantes spatiales pour CORSICA et temporelles pour PESTICA (voir section 2.2.2 pour la méthode) pour trouver les composantes liées au bruit physiologique et les enlever du signal mesuré.

4.2.1.2 Mouvements

L'analyse du mouvement a connu récemment un regain d'intérêt en raison de l'avènement de l'IRMf dont les résultats sont très sensibles aux mouvements spontanés du sujet. Power *et al.* ont proposé cinq indicateurs de mouvement fondés soit sur le déplacement des images, soit sur l'évolution du signal au cours de l'expérience : le *framewise displacement*,l'*absolute translational/rotational displacement*, la variation du signal dans des compartiments tissulaires et la moyenne du signal sur l'ensemble du cerveau (les deux premiers sont détaillés en section 5.3.1) [170].

Si une correction du mouvement apparaît nécessaire, il n'y a pas de consensus sur l'intérêt d'utiliser un facteur de correction plutôt qu'un autre. Siegel considère que le réalignement est indispensable mais n'est pas suffisant [187]. Car le réalignement ne corrige ni les modulations du signal ni les distorsions non linéaires de l'image qui résultent du mouvement. En effet, le réalignement rigide ne corrige ni les modulations du signal ni les distorsions non linéaires de l'image qui résultent du mouvement. Il sera donc insuffisant dans notre cas puisque Birn *et al.* [25] ont montré que les modulations du champ magnétique pendant la parole ou la déglutition peuvent générer des distorsions. Régresser le signal avec un modèle linéaire intégrant les paramètres de nuisance estimés au moment du réalignement des images peut réduire l'effet des mouvements de la tête mais pas les variations de susceptibilité ou les distorsions non rigides [228]. Ainsi, une correction non-linéaire du mouvement est nécessaire en plus du recalage rigide(ou réalignement). Friston *et al.* ont proposé une extension non-linéaire des paramètres de recalage rigide incluant la dérivée de chaque paramètre et l'élévation au carré des douze paramètres ainsi obtenus [71]. Caballero-Gaudes *et al.* suggèrent également d'utiliser le signal moyen dans la substance blanche ou dans le liquide céphalo-rachidien comme régresseur de non-intérêt.Cependant, ces signaux doivent être calculés avant tout lissage spatial et être définis sur des masques stricts pour éviter de mélanger des signaux de différents tissus [32].

Les approches les plus communément utilisées sont les approches de censure consistant à identifier et rejeter les images trop altérées à partir d'un ensemble de critères pour estimer le degré de mouvement ou la quantité de variations due aux artéfacts dans l'intensité de l'image. Siegel a comparé une censure avec ou sans augmentation (élimination des points précédents ou suivants le point éliminé par la censure) et les régresseurs de mouvements. Il a souligné la supériorité de la censure sur la régression, les deux méthodes permettant une augmentation de la significativité des activations. L'amélioration due à la régression semble modeste et insuffisante pour corriger

de fortes modulations du signal, comme en IRMf gustative. Il a également testé l'effet combiné de la censure et de la régression à partir de la dérivée des paramètres de mouvement, mais les changements de z-scores ne sont pas significativement différents de ceux observés en utilisant uniquement la censure. Cependant, il n'a étudié ni la régression par le signal global [173] ni la régression avec un signal extra-cérébral. En outre, Xu *et al.* considèrent les approches de censure comme problématiques en raison de la diminution de la puissance statistique, de la présence de données altérées qui ne dépasseraient pas le seuil de la censure et de la présence de séries temporelles discontinues plus difficiles à analyser. En effet, la censure réduit le nombre de degrés de liberté créant un déséquilibre dans le groupe puisque certains sujets qui présenteront plus de mouvement auront moins de degré de liberté. De plus, même si le signal est interpolé pour les scans censurés au lieu d'être supprimé, les discontinuités ou l'interpolation perturbent la corrélation temporelle du signal [32].

Une alternative à la censure et à l'utilisation des paramètres de mouvement est l'utilisation de méthodes conduites par les données. Caballero-Gaudes *et al.* expliquent en effet que les approches guidées par les données pourraient mieux tenir compte à la fois du mouvement entre les coupes et de l'hétérogénéité spatiale du bruit que la régression avec les paramètres de mouvements. Plusieurs auteurs ont créé des classifieurs à partir d'une décomposition des données en composantes indépendantes afin de séparer les composantes de signal des composantes de bruit [141, 207, 203, 228]. Ces classifieurs se fondent sur différents critères : temporels (corrélation avec les régresseurs des tâches exécutés ou avec les paramètres de mouvement), fréquentiels (proportion du spectre de puissance au-dessus d'un seuil donné, distribution dans les bandes de fréquence) et spatiaux (taille des clusters et leurs distributions, fréquence spatiale, entropie,la proportion de voxels dans chacun des tissus). Le classifieur choisi et le nombre de critères varient d'un classifieur à l'autre. Le débruitage à l'aide de l'analyse en composantes indépendantes est détaillée en section 2.2.2.3.

4.2.2 Parallélisation des algorithmes de clustering hirérachique

Olson a proposé un algorithme hiérarchique agglomératif parallèle pour plusieurs critères d'agglomération sur un PRAM (architecture à mémoire partagée) [159]. Rasmussen *et al.* ont développé l'algorithme de Ward sur un réseau de processeurs distribué ICL (qui fut le premier ordinateur massivement parallélisé commercialisé) [175]. Leur algorithme a atteint une accélération de 6 pour une matrice de taille $20 \times 10\,000$. Matias-Rodrigues *et al.* ont développé HPC_CLUST un clustering hiérarchique agglomératif parallèle sur un système à mémoire distribuée en utilisant l'interface MPI [142]. Ils ont testé leur algorithme sur un cluster de calcul de 24 nœuds avec 8 cœurs chacun pour parcelliser un million de séquences de gènes et ont ainsi calculé et trié la distance en 57 min au lieu de plus de 10 000 min avec un seul processeur. Du *et al.* ont implémenté une version MPI sur un cluster de nœuds en distribuant le calcul de la distance et également le calcul de la distance minimale et les mises à jour de la distance [59] sur les différents processeurs : ils ont obtenu une accélération maximale de 25 pour 48 processeurs pour une matrice $\boldsymbol{X}$ de taille $300 \times 10,000$.Dash *et al.* [54] ont proposé l'algorithme parallèle de recouvrement partiel dans lequel les données sont découpées en cellules qui se recouvrent partiellement, de sorte que plusieurs parcelles peuvent être fusionnées à la même itération s'ils appartiennent à des cellules différentes. La parallélisation consiste à distribuer les cellules sur les différents processeurs. Ils ont obtenu une accélération maximale de 8 avec 8 processeurs pour un jeu de données de taille $2 \times 60,000$. Zhang *et al.* ont quant à eux développé un algorithme hiérarchique agglomératif entièrement sur GPU pour la génétique en utilisant la mémoire texture du GPU et ont ainsi obtenu une accélération entre 2 et 4 pour tout l'algorithme en regroupant jusqu'à 4 096 gènes [233].

4.2.3 Parallélisation de la distance

Avec le développement des processeurs graphiques dédiés au calcul, plusieurs auteurs ont développé des algorithmes de calcul de la distance sur GPU. Chang *et al.* [40] ont proposé un noyau CUDA pour calculer la distance mais uniquement pour des matrices dont les dimensions sont multiples de 16. Leurs matrices de tests ont au plus n_V=12 000 vecteurs et n_T=64 caractéristiques, alors que les jeux de données plus larges on des centaines de milliers d'instances, la distance pour de tels jeux de données ne peut être calculée par cette méthode à cause de la taille limitée de la mémoire GPU. Kim *et al.* ont étendu le travail de Chang quelle que soit la dimension de la matrice en ajoutant des lignes et des colonnes de zéros à la matrice pour que les nouvelles dimensions soient multiples de 16 [117].Cependant, leur implémentation n'est pas adaptée pour les grands jeux de données à cause de la taille de la mémoire GPU. Li *et al.* ont proposé une implémentation du calcul de la distance en utilisant CUBLAS qui peut être étendu au calcul sur plusieurs GPUs [124]. Zhang *et al.* ont également parallélisé le calcul de la distance sur GPU mais en utilisant la mémoire texture et en prenant jusqu'à 4 096 gènes, ils ont accéléré le calcul de la distance d'un facteur 12 [233]. Sarje *et al.* ont développé un algorithme sur GPU pour le calcul de la distance entre toutes les paires ($\boldsymbol{M_{1}}_i$,$\boldsymbol{M_{2}}_i$) où $\boldsymbol{M_1}$ et $\boldsymbol{M_2}$ sont deux matrices contenant n_V vecteurs de taille n_T, adaptables à différentes distances [180]. Ils proposent de découper la matrice de distance en tuiles rectangulaires et de découper également ces tuiles en sous-tuiles afin de pouvoir réutiliser une partie des données que les threads du GPU ont chargé dans leur mémoire partagée. Lorsque la dimension n_T est trop grande pour qu'un vecteur entre en entier dans la mémoire partagée, ils décomposent le vecteur en coupes de dimension plus petite. Après une étude de la taille des tuiles et des coupes, ils obtiennent en simple précision une accélération autour de 350 et une accélération autour de 90 en double précision.Cependant, ils ont testé leur implémentation avec des vecteurs de grande dimension (n_T=5 419 et n_T=40 000) mais un nombre assez réduit de vecteurs (n_V variant entre 1 000 et 6 000). Avec un nombre peu élevé de vecteur, la matrice de distance tient dans la mémoire centrale du GPU et les vecteurs d'entrée sont découpés en coupes pour entrer dans la mémoire GPU. Plus récemment, Czarnul a implémenté le calcul de la distance en utilisant à la fois OpenMP et CUDA [52]. Pour cela, il a découpé la matrice de triangulaire en blocs rectangulaires qui sont ensuite réparti entre le GPU et le CPU. Il affecte un processus léger pour chaque GPU disponible afin de distribuer le calcul sur le GPU et il affecte les processus légers restants au calcul de la distance. Pour un jeu de données de taille n_T=20 000 et n_V=10 000, il a obtenu une accélération de 5.

Plusieurs parallélisations pour le calcul de la distance euclidienne et l'algorithme de clustering hiérarchique agglomératif ont été proposés. Cependant, excepté l'algorithme proposé par Matias-Rodrigues *et al.*, aucune implémentation n'a été appliquée à la parcellisation d'un grand nombre de vecteurs avec un nombre de caractéristiques assez grand. Pour pouvoir traiter les données d'IRMf qui comportent entre 50 000 et 500 000 vecteurs de dimension entre 100 et 1 000, il apparaît nécessaire de développer de nouvelles implémentations de l'algorithme de Ward.

4.3 Organisation du manuscrit

Le chapitre 5 décrit le jeu de données acquis pour la thèse ainsi que l'analyse des perturbations et leur correction. Le chapitre 6 présente l'algorithme de Ward et sa parallélisation. Le chapitre 7 contient le travail réalisé pour le calcul de la distance : algorithmes séquentiel et parallèles, ainsi que le stockage de la matrice de distance. Le chapitre 8 donne les résultats numériques pour le calcul de la distance et la parallélisation de l'algorithme de Ward. Le chapitre 9 illustre l'application de la parcellisation au jeu de données. Enfin, le chapitre 10 conclut la thèse.

Deuxième partie

Données&Méthodes

Chapitre 5

Données

Ce chapitre présente le jeu de données acquis pour la thèse dans la première section et les prétraitements appliqués, en particulier la méthode de normalisation, des données dans la deuxième section. La troisième section est consacrée à la caractérisation du mouvement et sa correction par la méthode de l'ellipse de confiance. La dernière partie explique la manière dont les données sont parcellisées.

5.1 Jeu de données

5.1.1 Participants

Avant le recrutement pour la session d'IRMf, les participants ont été invités à une session préalable. Chaque participant a dû confirmer qu'il n'avait pas d'antécédents cliniques de graves maladies et qu'il disposait de capacités olfactives et gustatives normales. Pour détecter les insuffisances olfactives ou gustatives, chaque participant a passé le test européen des capacités olfactives (ETOC) [204] ainsi qu'un test de reconnaissance des quatre goûts basiques (c'est-à-dire sucré, salé, acide et amer). Les participants ont ensuite attribué un score à la boisson utilisée lors de l'expérience d'IRMf sur une échelle analogue visuelle de dix centimètres avec le label « Je n'aime pas du tout » placé à gauche au point 0 cm et le label « J'aime beaucoup » placé à droite au point 10 cm. Les résultats montrent que les participants ont évalué positivement la boisson $(6,4 \pm 1,6)$.

Les participants se sont également entraînés à boire et à avaler dans les conditions de l'expérience lors d'une session dédiée une semaine avant l'acquisition. Lors de cette session, les participants étaient couchés sur leur dos, la tête immobilisée par une fausse bobine de tête et les stimuli étaient délivrés par un gustomètre compatible avec le scanner et aux mêmes temps que lors de l'acquisition. Quatre participants sains, droitiers et volontaires (un homme et trois femmes), âgés de $23,2 \pm 1,5$ ans et d'indice de masse corporelle $22,4 \pm 2,3\ kg/m^2$, ont participé à l'étude. Les participants n'ont pas été autorisés à manger et n'ont pu boire que de l'eau pendant les deux heures précédant la session d'IRMf. Cette étude a été approuvée par le comité éthique de l'hôpital universitaire de Clermont-Ferrand (AU1283) et l'accord écrit de chaque participant a été obtenu conformément à la déclaration d'Helsinki.

5.1.2 Paradigme

Le paradigme fonctionnel est découpé en blocs. Une répétition consiste en quatre phases successives qui commencent par un signal visuel d'avertissement pendant deux secondes, suivi de la réception d'un millilitre d'eau de source (Volvic, Danone, France) ou d'une boisson sucrée (sirop de pêche dilué à 10,2 g/100 mL d'eau de source, Teisseire, France) pendant cinq secondes, liquide gardé en bouche pendant cinq secondes puis la phase de déglutition pendant trois secondes. La phase de repos suivante dure vingts secondes plus ou moins un jitter aléatoire (d'une durée

maximale de 1,5 secondes). Chaque phase est signifiée au sujet au moyen d'une image particulière (**cf. Figure 5.1**. Ce bloc de trente-cinq secondes a été répété quarante fois par session. Les deux boissons ont été présentées alternativement. Deux sessions de vingts-quatre minutes ont été réalisées pour chaque sujet pendant la même session.

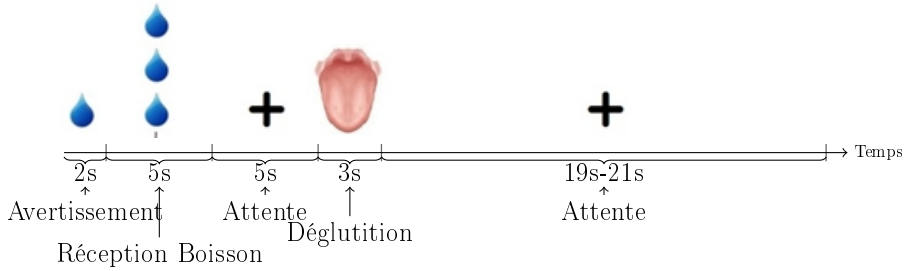


FIGURE 5.1 – Paradigme de l'expérience de déglutition.

Tous les stimuli liquides ont été administrés à température ambiante ($23°C$) avec un gustomètre. Le gustomètre est composé de huit pompes à seringues programmables (Aladin Pump, World Precision Instrument, Sarasota, États-Unis), contrôlées par le logiciel E-Prime 2.0 (Psychological Software Tools, Sharpsburg, États-Unis). Chaque pompe tient une seringue en verre d'une contenance de 50 mL (Fortuna Optima, Poulten & Graf GmbH, Wertheim, Allemagne) connectée à un tube de distribution de boisson Tygon®(Saint-Gobain Performance Plastics, France). Les huit tubes de distribution convergent vers le même collecteur, fixé à la bobine de tête, de manière à obtenir une administration reproductible de la solution sur la langue des sujets [215]. Les signaux visuels ont été présentés sur un écran via un système de stimulation visuelle compatible avec le scanner (NordicNeuroLab, Bergen, Norvège).

Lors de cette expérience, un capteur de pression a aussi été posé sous l'épiglotte, afin de détecter l'ouverture de la bouche et la déglutition du sujet. Le relevé de pression est étudié en section 9.1.

5.1.3 Acquisition des données

Les données d'imagerie ont été acquises sur un système à résonance magnétique General Electric Discovery MR750 à 3 T (General Electric Medical Systems, Milwaukee, États-Unis). Une bobine de tête à 32 canaux *receive-only phased-array* a été utilisée pour l'acquisition des images cérébrales. Une séquence (BRAVO) d'acquisition FSPGR 3D avec préparation IR a été utilisée pour acquérir les images anatomiques du cerveau entier pondérées en T1 pour chaque sujet. Les images T2*-pondérées à gradient d'écho ont été acquises en utilisant un encodage 2D et un seul volume utilisant une trajectoire cartésienne de type Echo-Planar ou EPI (TR=3000 ms, TE=30 ms et angle d'inclinaison=90°). Les images ont été acquises en haute-résolution spatiale selon une orientation axiale coplanaire avec la ligne CA-CP (commissure antérieure-commissure postérieure) avec un petit volume de voxels égal à 1,5 × 1,5 × 1,8 mm^3 (42 coupes contiguës entrelacées, taille=192 × 192 mm^2, une matrice de 128 par 128, une bande-passante en réception de 250 kHz et encodage de la phase de la gauche vers la droite). Une haute résolution spatiale a été choisie afin de faciliter la détection des faibles activations en particulier dans les régions qui tendent à être sensibles aux artéfacts de susceptibilité [106].

Les images fonctionnelles acquises pendant l'expérience sont fortement entachées de distorsions géométriques, notamment sur les coupes inférieures. Ces distorsions sont dues à l'utilisation la technique EPI qui favorise la rapidité au prix d'une sensibilité accrue aux hétérogénéités du champ magnétique. Pour faciliter la correction a posteriori de ces distorsions, une image de référence présentant sensiblement le même contraste mais sans distorsion géométrique a été acquise pour chaque sujet. Cette image pondérée en T2 (notée T2w) s'ajoute à l'imagerie pondérée T1 pour constituer les acquisitions anatomiques permettant de normaliser les images fonctionnelles.

5.2 Prétraitements

Tous les prétraitements ont été effectués à l'aide du logiciel SPM12 (Statistical Parametric Mapping ; Welcome Trust Dept of Cognitive Neurology, Londres, Royaume-Uni) disponible sous MATLAB. Les images suivent un processus de prétraitement classique en IRMf de slice timing correction, de recalage des images fonctionnelles sur les images anatomiques et de normalisation dans un espace neuroanatomique de référence (espace MNI). Classiquement, les images EPI sont recalées sur l'image anatomique T1 au moyen d'une transformation rigide. Notre originalité réside essentiellement dans l'idée de se servir de l'image T2w pour effectuer une étape de correction des distorsions géométriques à l'échelle individuelle. Pour cela, la boîte à outils de DARTEL (*Diffeomorphic Anatomical Registration using Exponentiated Lie algebra* [14]) a été utilisée car elle dispose de fonctions de recalage non linéaires.C'est pourquoi nous ne détaillons que ce point et nous contentons de donner les paramètres utilisés pour les autres étapes.

5.2.1 Prétraitements classiques

Tout d'abord, les volumes EPIs de chaque sujet sont corrigés pour le *slice timing* avec les paramètres suivants : nombre de coupes=42, ordre des coupes croisé c'est-à-dire coupes impaires puis coupes paires, coupe de référence : la coupe n^o 2, donnant les volumes aEPI. L'étape suivante est le réalignement (Estimation&Reslice) : les EPIs sont réalignées sur l'image moyenne calculée pour les deux sessions. L'estimation est calculée avec les paramètres suivants : qualité de 1, séparation fixée à 1,5 avec un smoothing isotrope de 3 mm sans *warping* et en interpolant avec des B-splines d'ordre 7. Le *reslice* s'effectue sans *warping* et en interpolant avec des B-splines d'ordre 7. À la fin de l'étape de réalignement et de *reslice* (boîte Réalignement), on obtient des images réalignées (raEPIs) et leur image moyenne ($\overline{raEPI}$). Vient ensuite l'étape de *coregistration*. Afin d'améliorer les résultats de la *coregistration*, nous définissons l'origine de l'EPI moyenne, des EPIs, de la T1 pondérée et de la T2 pondérée sur la commissure antérieure. Il faut d'abord rééchantilloner la T1 pondérée pour obtenir la même résolution spatiale qu'en EPI (1,5 mm isotrope). Nous commençons par coregistrer la T2 pondérée (image source) sur la T1 pondérée (image de référence), ce qui permet d'unifier la géométrie entre la T1 pondérée et la T2 pondérée et d'effectuer un premier recalage affine (ce qui donne une image T2w coregistrée notée cT2w). L'EPI moyenne (image source) est ensuite coregistrée sur la T2 pondérée (image de référence) obtenue par la coresgistration précédente : l'image moyenne coregistrée notée $\overline{craEPI}$ et une transformation affine qui est appliquée aux EPIs permettant d'obtenir les EPIs coregistrées (craEPI). Les deux *coregistrations* se font avec les paramètres par défaut de SPM excepté pour le vecteur de séparation qui vaut [4 2 1,5] et l'interpolation qui se fait avec des B-splines d'ordre 7.

5.2.2 Normalisation

5.2.2.1 Méthode

Après recalage des deux images anatomiques (T1w et T2w), l'EPI moyenne (image source) est recalée sur la T2 pondérée (image de référence) avec une transformation affine. Les corrections appliquées jusqu'ici sont seulement affines. L'idée est donc d'appliquer la fonction de recalage non-linéaire de DARTEL. Celle-ci ne traitant que des images binaires, l'EPI moyenne et les images anatomiques sont segmentées avec la fonction Segment avec les paramètres par défaut. Cette segmentation génère une carte floue d'appartenance aux différentes classes de tissus cérébraux ; substance grise (préfixe c1 et rc1), substance blanche (préfixe c2 et rc2) et liquide céphalo-rachidien (préfixe c3 et rc3). Les flots de déformation (u_rc_$\overline{EPI}$ et u_rc_T1w) sont ensuite calculés par la fonction *Run DARTEL* : en regroupant le tissu de substance grise issu de la segmentation de la T1 pondérée avec celui issu de la segmentation de l'EPI moyenne coregistrée ; le tissu de substance blanche issu de la segmentation de la T1 pondérée avec celui issu de la

segmentation de l'EPI moyenne coregistrée ; et le tissu du liquide céphalo-rachidien issu de la segmentation de la T1 pondérée avec celui issu de la segmentation de l'EPI moyenne coregistrée, en utilisant les paramètres par défaut (**cf. Section A.1**). Cette transformation non-linéaire (appelée également étape de *warping*) est appliquée à toutes les images EPI, sujet par sujet.

Ensuite l'étape de normalisation DARTEL classique (étapes 8 et 9) est appliquée. Elle consiste à construire un modèle moyen après recalage non-linéaire inter-sujet des images anatomiques T1w. Ce modèle est ensuite recalé à un modèle moyen qui est positionné dans le repère neuroanatomique de référence MNI. Ces deux étapes sont appliquées bien sûr sur l'ensemble des acquisitions T1.

5.2.2.2 Résultats

Pour comparer l'approche conventionnelle (une seule normalisation au niveau du groupe) et notre approche, les images cmeans (EPIs moyennes coregistrées) ont été moyennées sur les quatre sujets. La figure 5.3 montre les images moyennes avec un *warping* individuel à gauche et sans *warping* individuel à droite (approche conventionnelle). Pour les coupes inférieures, l'ajout de l'étape de *warping* individuel permet de retrouver du signal (zones grises sur les figures de gauche correspondant à des zones noires sur les figures de droite).

L'amélioration induite par le *warping* individuel est spectaculaire, notamment au niveau des coupes inférieures qui présentent deux améliorations majeures : (1) les contours du cerveau sont plus nets, ce qui illustre une meilleure correspondance anatomique inter-sujets, (2) les atténuations du signal dues aux différences de susceptibilité magnétique sont partiellement compensées par le *warping* individuel. C'est donc cette méthode, plus complexe, mais plus performante qui est utilisée.

5.3 Caractérisation du mouvement et correction

5.3.1 Notations et définitions

Le tableau 5.1 définit les différents paramètres de mouvement utilisés par la suite. Pour les dérivées des paramètres de réalignement, on considère par convention la valeur au premier pas de temps comme égale à 0. TAD est l'abréviation de *Translational absolute displacement*. RAD est l'abréviation de *Rotational absolute displacement*. FD est l'abréviation de *framewise displacement*. Les notations TAD' et RAD' sont abusives car TAD' (respectivement RAD') n'est pas la dérivée de $TAD(t)$ (respectivement RAD) mais la somme des dérivées.

5.3.2 Quelques mesures caractérisant les données

La caractérisation du mouvement du sujet repose d'abord sur les paramètres issus du recalage rigide effectué pour mettre en correspondance toutes les images EPI acquises au cours du temps. Les paramètres obtenus quantifient donc le mouvement de tête du sujet entre deux acquisitions successives espacées de TR (ici 3 s). Ces paramètres ne reflètent que partiellement le mouvement effectué au cours du paradigme, en particulier les mouvements non rigides (ouverture de bouche, déglutition, ...).

Le tableau 5.2 montre les statistiques des paramètres issus du recalage rigide

$$[\Delta x(t) \quad \Delta y(t) \quad \Delta z(t) \quad roll(t) \quad pitch(t) \quad yaw(t)] \ .$$

Le tableau montre pour chacun la valeur moyenne (μ), l'écart-type (σ), la médiane ($Md.$) et les extrema (Min et Max). Les paramètres de mouvement montrent une variabilité intra-sujet : les moyennes changent entre les deux sessions. De plus, il y a également de la variabilité inter-sujet (en comparant par exemple le paramètre $roll(t)$ pour les sujets n^{os} 1 et 2). **En se fondant sur les caractéristiques du paramètres du mouvement, on peut en déduire qu'un soin**

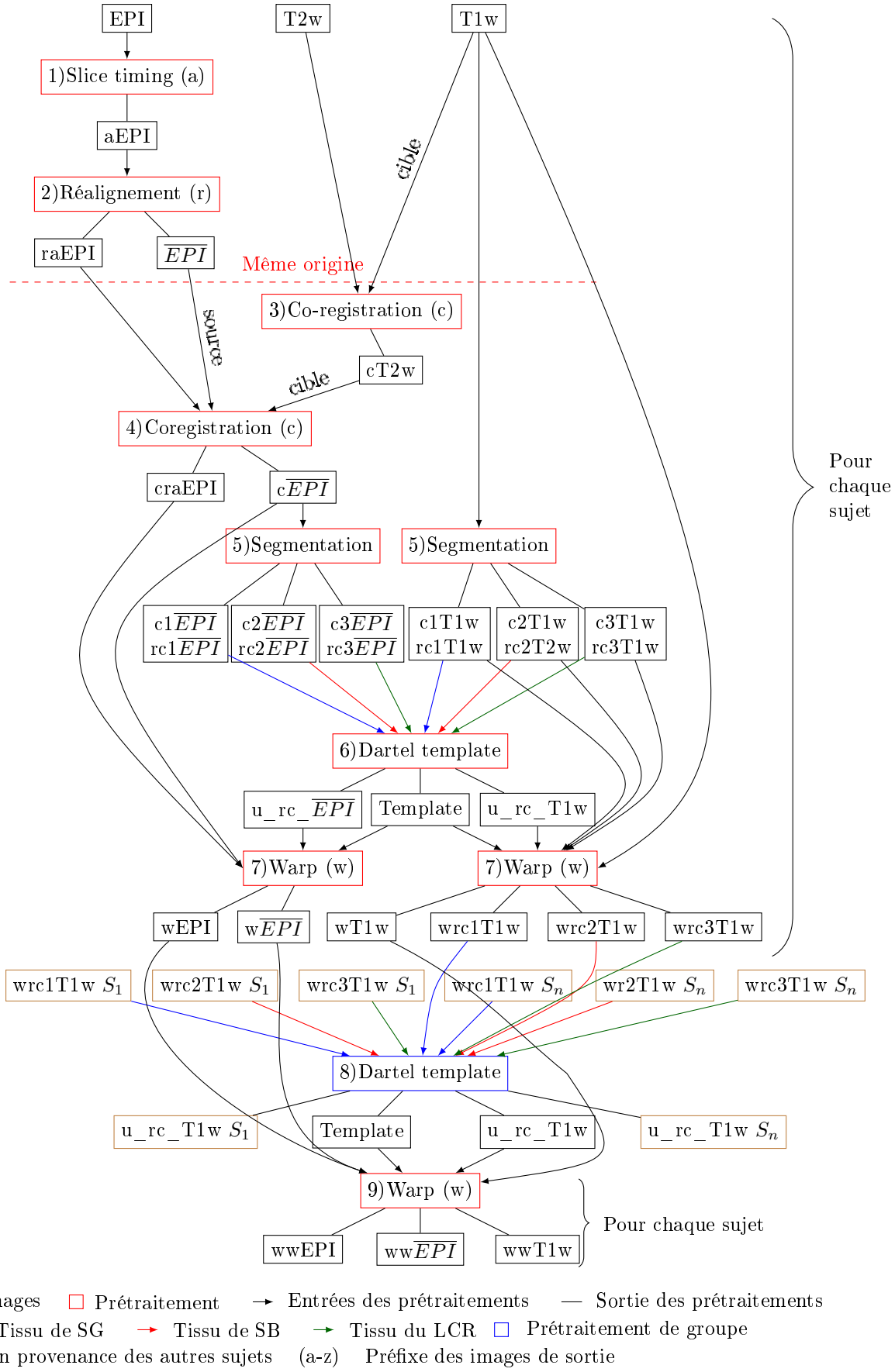
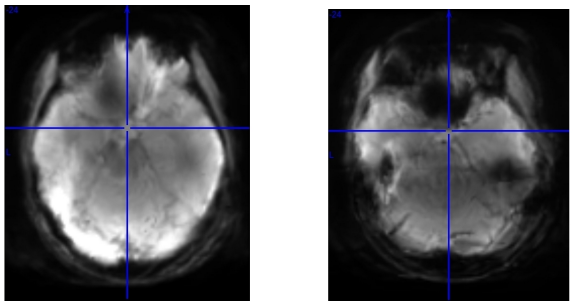
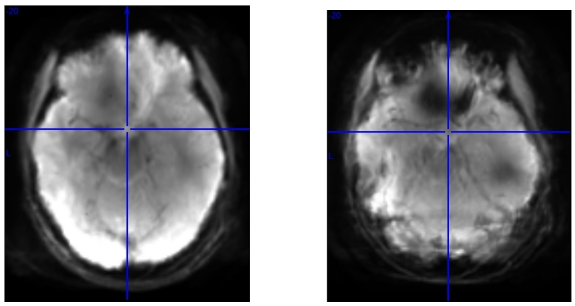


FIGURE 5.2 – Chaîne des prétraitements : Les EPIs subissent les prétraitements classiques (étapes 1 à 5). Une étape de *warping* (c'est-à-dire de déformation non linéaire) est ajoutée à l'échelle individuelle (étapes 6 à 7). Après ce *warping* individuel, la normalisation de groupe est appliquée (étapes 8 à 9).

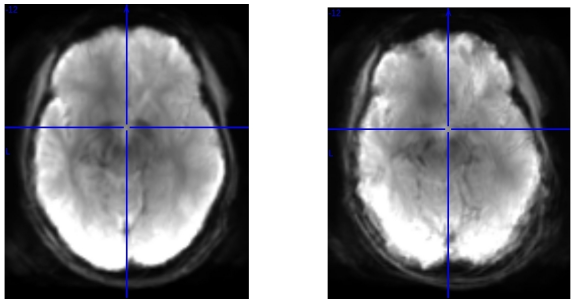
La segmentation donne les cartes de tissus de la substance grise (SG, préfixe c1), de la substance blanche (SB, préfixe c2) et du liquide céphalo-rachiden (LCR, préfixe c3). Les images rc* correspondent aux cartes de tissus « importés » pour DARTEL.



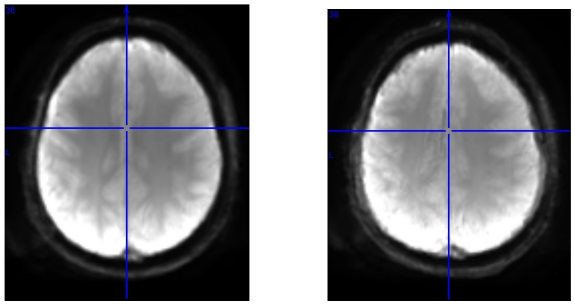
(a) Avec *warping* individuel (b) Sans *warping* individuel



(c) Avec *warping* individuel (d) Sans *warping* individuel



(e) Avec *warping* individuel (f) Sans *warping* individuel



(g) Avec *warping* individuel (h) Sans *warping* individuel

FIGURE 5.3 – Comparaisons de plusieurs coupes axiales pour les deux approches : figures a) et b) z=-24 mm MNI ; figures c) et d) z=-20mm MNI ; figure e) et f) z=-12 mm MNI et figures g) et h) z=38 mm MNI.

Paramètres de réalignement	
$\Delta x(t)$	Translation
$\Delta y(t)$	Translation
$\Delta z(t)$	Translation
$roll(t)$	Rotation
$pitch(t)$	Rotation
$yaw(t)$	Rotation
Indicateurs de mouvement	
TAD (t)	$\lvert\Delta x(t)\rvert + \lvert\Delta y(t)\rvert + \lVert\Delta z(t)\rvert$
RAD (t)	$\lvert roll(t)\rvert + \lvert pitch(t)\rvert + \lvert yaw(t)\rvert$
Dérivées de paramètres de réalignement	
$x'(t)$	$\Delta x(t) - \Delta x(t-1)$
$y'(t)$	$\Delta y(t) - \Delta y(t-1)$
$z'(t)$	$\Delta z(t) - \Delta z(t-1)$
$roll'(t)$	$roll(t) - roll(t-1)$
$pitch'(t)$	$pitch(t) - pitch(t-1)$
$yaw'(t)$	$yaw(t) - yaw(t-1)$
Indicateurs de mouvement	
FD (t)	$\lvert x'(t)\rvert + \lvert y'(t)\rvert + \lvert z'(t)\rvert + \lvert roll'(t)\rvert + \lvert pitch'(t)\rvert + \lvert roll'(t)\rvert$
TAD' (t)	$\lvert x'(t)\rvert + \lvert y'(t)\rvert + \lvert z'(t)\rvert$
RAD' (t)	$\lvert roll'(t)\rvert + \lvert pitch'(t)\rvert + \lvert roll'(t)\rvert$
Variations du signal	
s_{WM}	Variation moyenne dans la substance blanche (masque à 90%) $s_{WM} = \sum_{v \in WM} \frac{s(v,t) - \overline{s(v)}}{\overline{s(v)}} \times 100$
s_{GM}	Variation moyenne dans la substance grise (masque à 90%) $s_{GM} = \sum_{v \in GM} \frac{s(v,t) - \overline{s(v)}}{\overline{s(v)}} \times 100$
MP (t)	Ensemble des paramètres définis ci-dessus

TABLE 5.1 – Paramètres de mouvement.

particulier doit être apporté au mouvement puisqu'il y a à la fois une variabilité intra-sujet et une variabilité inter-sujet. Quant au relevé de pression, les statistiques ne diffèrent pas beaucoup entre les sujets excepté le minimum du relevé de pression.

5.3.3 Mise en évidence de perturbations dans la substance grise

Les diagrammes de Power ont été introduit pour caractériser le mouvement à l'échelle individuelle et pour évaluer la qualité des données [169]. Leur idée est de mettre en correspondance les paramètres de recalage rigide avec la variation de signal dans la substance grise et dans la substance blanche. Ils ont été introduits comme un outil d'évaluation de la qualité des données acquises car ils permettent d'identifier différents artéfacts. Utilisés en IRMf au repos, nous proposons de l'étendre aux données d'IRMf avec tâche. En effet, ces diagrammes sont complémentaires à l'étude menée sur les paramètres de réalignement car ils permettent de visualiser les conséquences du mouvement sur le signal.

Les diagrammes de Power [169] tracés pour chacune des sessions d'acquisition de l'expérience de déglutition présentée en section 5.1 pour les quatre sujets (**cf. Figures 5.4 à 5.7**) mettent en évidence la présence de fortes perturbations dans la substance blanche, perturbations se propageant dans la substance grise. Ces modulations dans la substance blanche sont inattendues et sont donc le stigmate d'un artéfact de mouvement. Les sessions ont été séparées car elles ne présentent pas la même ligne de base. La variation du signal a été calculée par rapport à la ligne de base caractéristique de la session d'acquisition. Les figures 5.4 à 5.7 montrent les diagrammes de Power pour les quatre sujets. Le premier graphique représente l'évolution de TAD' au cours du temps, le deuxième graphique montre l'évolution de RAD' au cours du temps. Le troisième

Paramètres de mouvement		Sujet 1		Sujet 2		Sujet 3		Sujet 4	
		Session 1	Session 2	Session 1	Session 2	Session 1	Session 2	Session 1	Session 2
$\Delta x(t)$	μ	-0.15	-2.99	0.11	-1.24	0.43	-1.72	0.08	-2.68
	σ	0.53	0.45	0.08	0.06	0.17	0.16	0.09	0.06
	Méd.	-0.34	-2.77	0.11	-1.24	0.46	-1.73	0.07	-2.68
	Min	-0.86	-5.22	-0.09	-1.44	-0.01	-2.14	-0.13	-2.82
	Max	1.31	-2.21	0.33	-1.09	0.94	-1.27	0.36	-2.49
$\Delta y(t)$	μ	-1.63	-2.52	0.19	0.28	-0.40	0.24	0.06	0.23
	σ	0.64	0.42	0.11	0.06	0.40	0.23	0.05	0.04
	Méd.	-1.64	-2.43	0.19	0.29	-0.34	0.23	0.08	0.23
	Min	-2.98	-3.43	-0.16	0.13	-1.13	-1.00	-0.21	0.03
	Max	0.21	-1.51	0.43	0.46	0.45	0.84	0.17	0.32
$\Delta z(t)$	μ	-0.19	-1.05	0.00	-0.57	0.49	0.90	0.82	0.75
	σ	0.26	0.22	0.19	0.31	0.32	0.34	0.35	0.15
	Méd.	-0.13	-1.06	0.01	-0.70	0.45	0.94	0.86	0.74
	Min	-1.27	-1.66	-0.58	-1.11	-0.36	-0.32	-0.32	0.17
	Max	0.74	-0.45	0.64	0.15	1.52	1.87	2.05	1.75
$roll(t)$	μ	-1.46	-2.26	-0.12	-0.04	0.95	0.36	-0.55	-0.14
	σ	0.58	0.36	0.31	0.26	0.50	0.30	0.33	0.21
	Méd.	-1.48	-2.17	-0.13	-0.03	0.91	0.35	-0.53	-0.15
	Min	-1.27	-1.66	-0.58	-1.11	-0.36	-0.32	-0.32	0.17
	Max	0.17	-1.31	0.90	0.58	1.89	1.99	0.16	0.36
$pitch(t)$	μ	-0.33	-1.11	-0.14	-0.03	-0.22	-0.95	0.05	-0.23
	σ	0.31	0.22	0.08	0.08	0.15	0.23	0.17	0.08
	Méd.	-0.45	-1.05	-0.12	-0.03	-0.23	-0.95	0.08	-0.23
	Min	-0.73	-2.09	-0.40	-0.20	-0.59	-1.54	-0.30	-0.43
	Max	0.44	-0.58	0.06	0.17	0.12	-0.47	0.47	-0.02
$yaw(t)$	μ	0.05	-0.50	0.07	0.14	0.28	-0.64	-0.26	0.13
	σ	0.11	0.32	0.05	0.04	0.09	0.28	0.14	0.07
	Méd.	0.06	-0.39	0.07	0.15	0.28	-0.54	-0.27	0.14
	Min	-0.25	-2.48	-0.09	-0.02	-0.01	-1.34	-0.68	-0.28
	Max	0.30	-0.12	0.24	0.27	0.61	-0.18	0.00	0.28

TABLE 5.2 – Quelques statistiques sur les paramètres de mouvement.

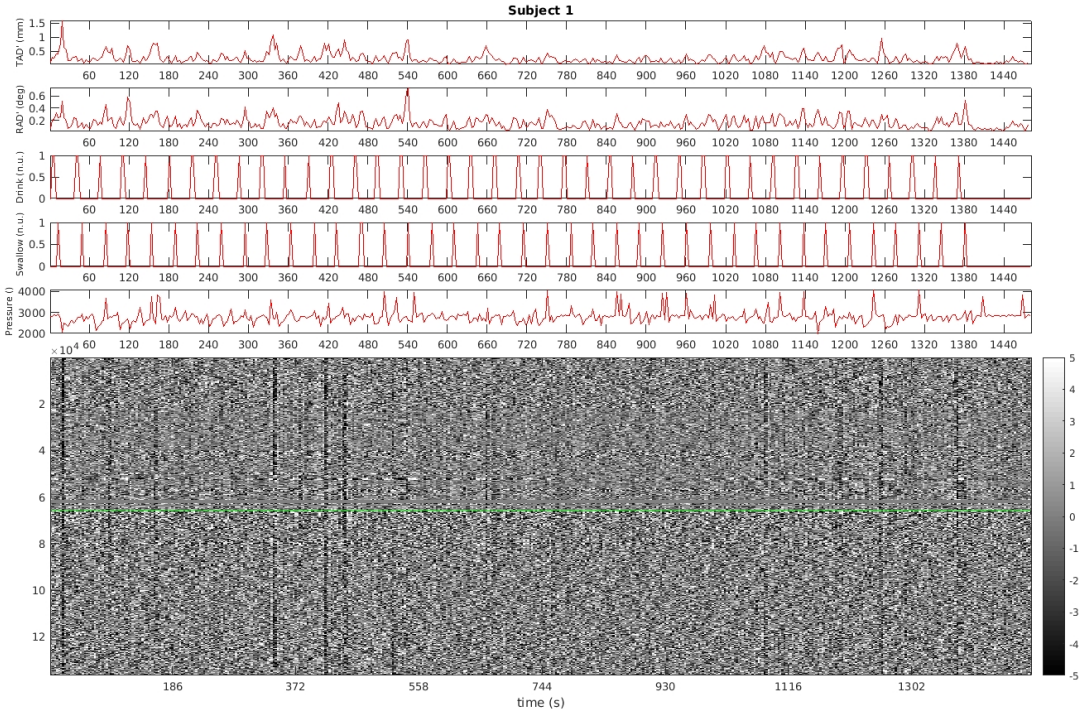
graphique correspond aux périodes de réception de boisson (d'une durée de 3 secondes) et le quatrième graphique correspond aux périodes de déglutition (d'une durée de 3 secondes). Le cinquième graphique correspond au relevé de pression mesuré au niveau de la glotte. La carte en niveaux de gris représente les variations du signal mesuré dans la substance grise (masque à 90% **cf. Section A.1**, partie au-dessus de la ligne verte) et dans la substance blanche (masque à 90%, partie en dessous de la ligne verte).

Les huit diagrammes de Power présentés aux figures 5.4 à 5.7 mettent en évidence que la variabilité inter-session et la variabilité inter-individuelle observées à l'aide des paramètres de mouvement se retrouvent dans les variations de signal. Certaines sessions sont en effet presque exemptes de fortes variations négatives se propageant de la substance blanche dans la substance blanche. De plus, si ces variations sont essentiellement négatives, certaines variations sont positives (session n^o 2 du sujet 2). Pour certains sujets, les variations de signal sont contenues dans une seule partie de la substance blanche. Nous remarquons également le cas de la session n^o 1 du sujet 4 qui contient des perturbations continues sur tout un intervalle de temps.

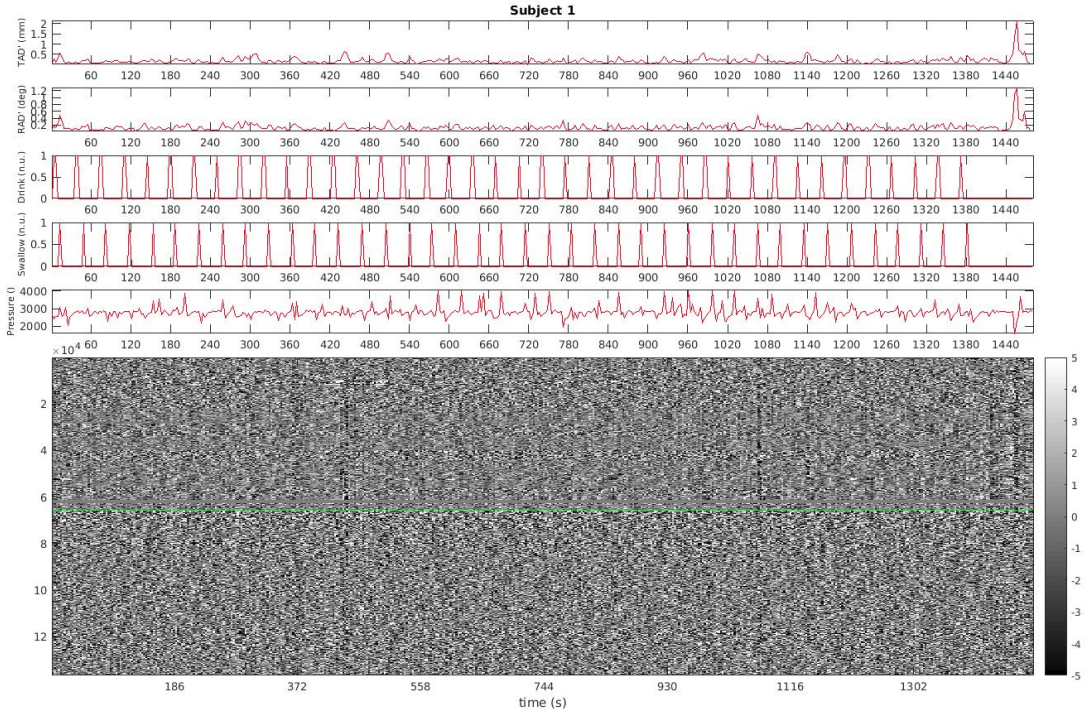
Les diagrammes de Power sont un moyen élégant pour caractériser finement (c'est-à-dire à l'échelle individuelle et au cours du paradigme) l'influence du mouvement sur le signal image. Des variations importantes du signal sont clairement mises en évidence. Elles risquent de masquer des activations ou de créer de fausses activations. Il sera donc nécessaire de les considérer comme perturbations dans l'analyse. Avant d'aborder ce point, les parties suivantes visent à caractériser l'origine des perturbations.

5.3.4 Ces variations du signal sont-elles liées au paradigme ?

Pour déterminer si ces variations sont dues au paradigme, une inspection visuelle a été menée afin d'identifier les variations présentes à la fois dans la substance blanche et dans la substance

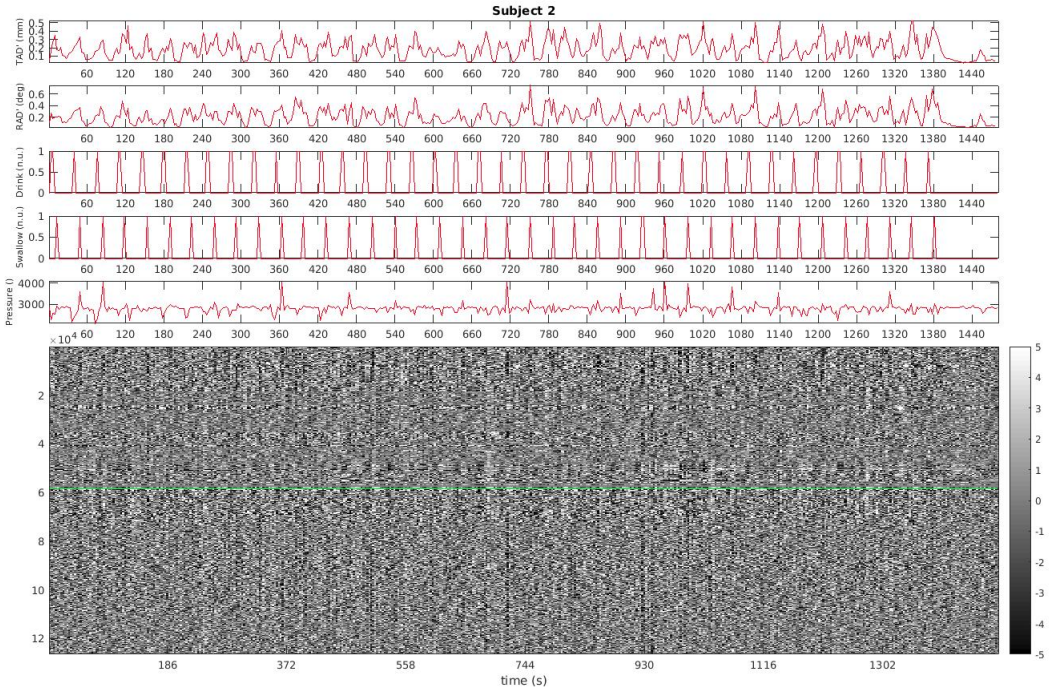


(a) Session 1

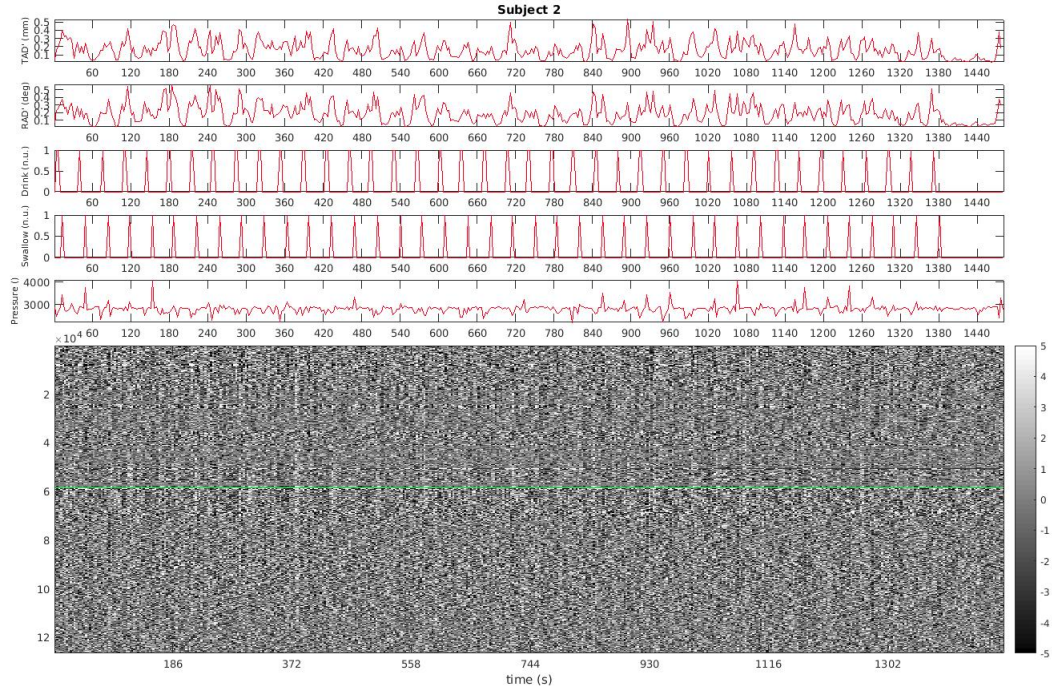


(b) Session 2

FIGURE 5.4 – Diagramme de Power pour le sujet n^o 1.
Le premier graphique correspond à l'évolution de $TAD'(t)$ en mm. Le deuxième graphique correspond à l'évolution de $RAD'(t)$ en deg. La troisième courbe représente la phase de réception de la boisson du paradigme présenté en figure 5.1. La quatrième courbe représente la phase de déglutition du paradigme présenté en figure 5.1. Le cinquième graphique représente le signal de pression mesuré sous l'épiglotte. La carte de gris correspond à la variation de signal par rapport à la ligne de base mesuré dans les voxels de la substance grise et de la substance blanche. La ligne verte marque la séparation entre la substance grise (en haut) et la substance blanche (en bas).

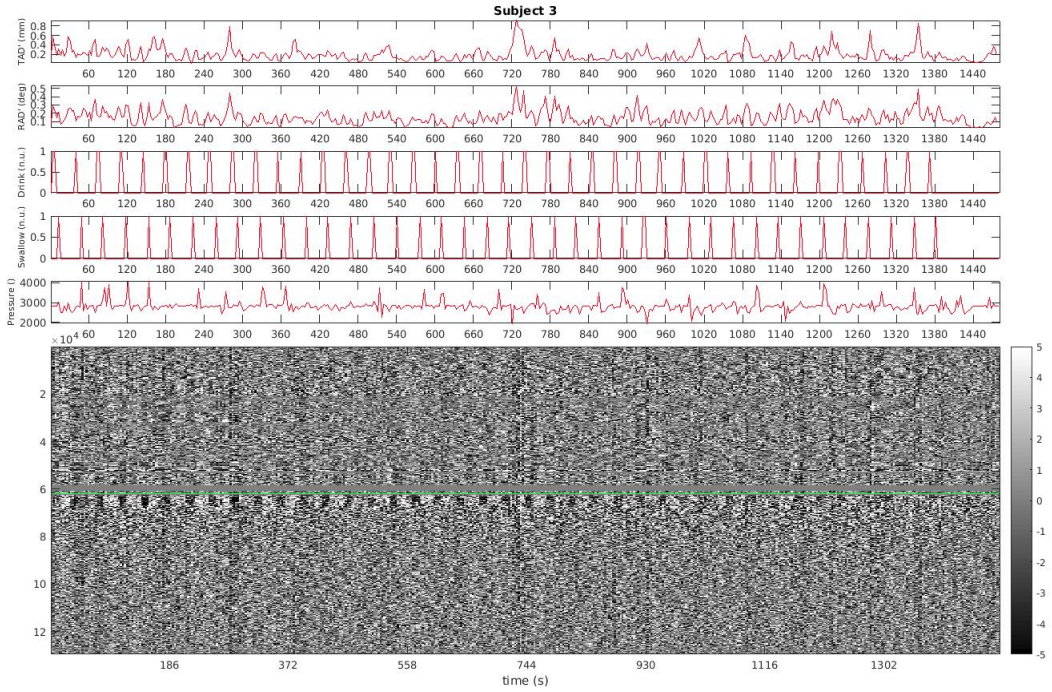


(a) Session 1

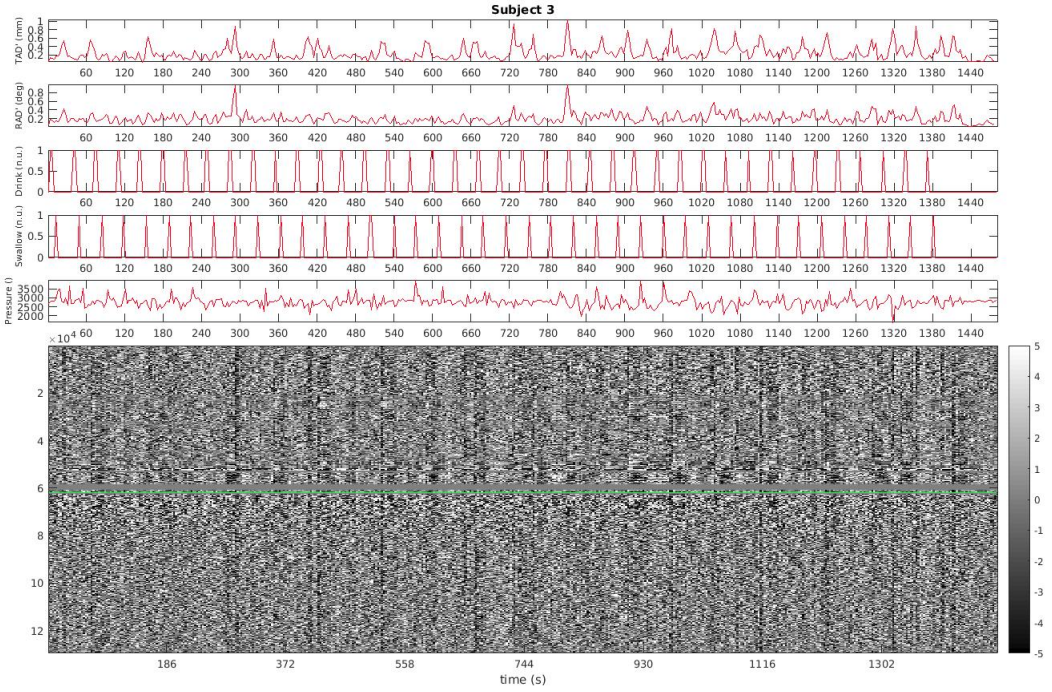


(b) Session 2

FIGURE 5.5 – Diagramme de Power pour le sujet n^o 2.
Le premier graphique correspond à l'évolution de $TAD'(t)$ en mm. Le deuxième graphique correspond à l'évolution de $RAD'(t)$ en deg. La troisième courbe représente la phase de réception de la boisson du paradigme présenté en figure 5.1. La quatrième courbe représente la phase de déglutition du paradigme présenté en figure 5.1. Le cinquième graphique représente le signal de pression mesuré sous l'épiglotte. La carte de gris correspond à la variation de signal par rapport à la ligne de base mesuré dans les voxels de la substance grise et de la substance blanche. La ligne verte marque la séparation entre la substance grise (en haut) et la substance blanche (en bas).

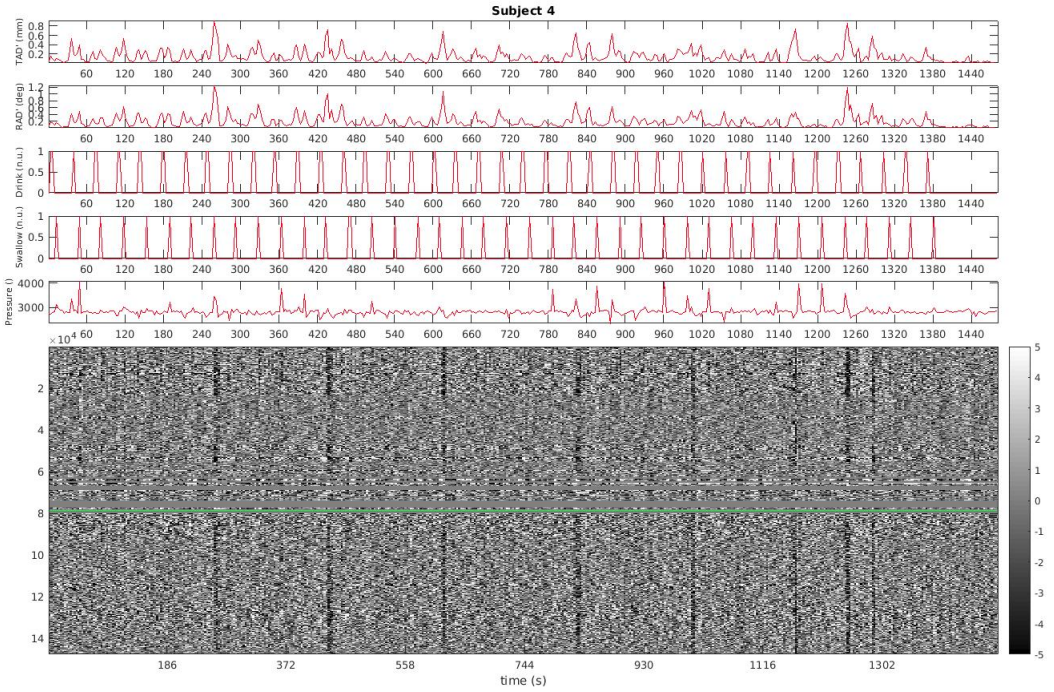


(a) Session 1

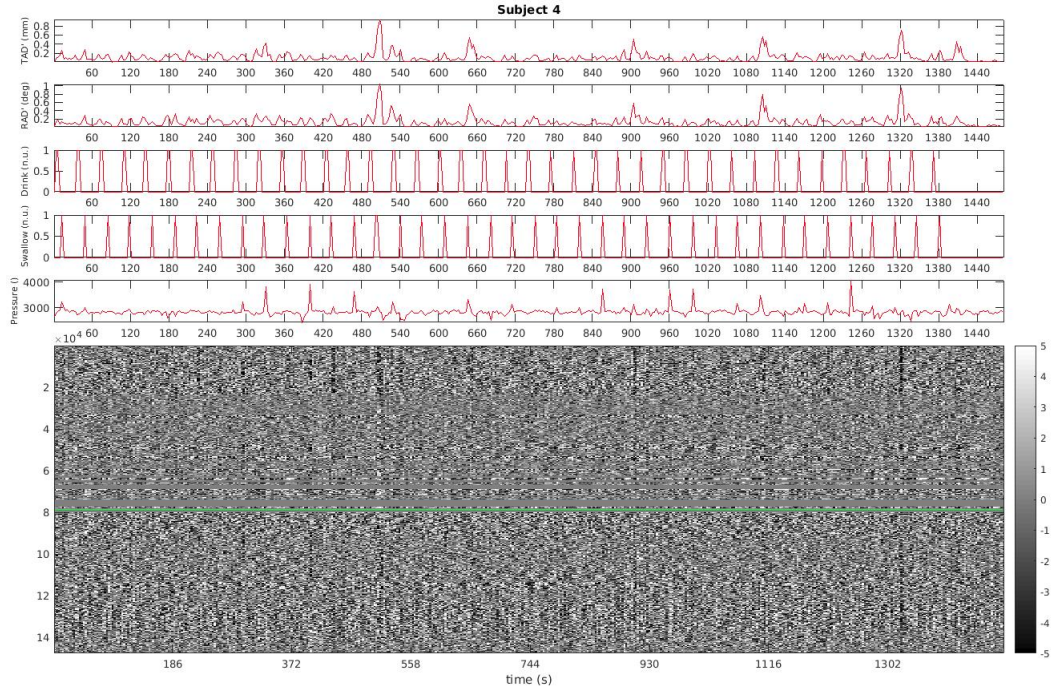


(b) Session 2

FIGURE 5.6 – Diagramme de Power pour le sujet n^o 3.
Le premier graphique correspond à l'évolution de $TAD'(t)$ en mm. Le deuxième graphique correspond à l'évolution de $RAD'(t)$ en deg. La troisième courbe représente la phase de réception de la boisson du paradigme présenté en figure 5.1. La quatrième courbe représente la phase de déglutition du paradigme présenté en figure 5.1. Le cinquième graphique représente le signal de pression mesuré sous l'épiglotte. La carte de gris correspond à la variation de signal par rapport à la ligne de base mesuré dans les voxels de la substance grise et de la substance blanche. La ligne verte marque la séparation entre la substance grise (en haut) et la substance blanche (en bas).



(a) Session 1

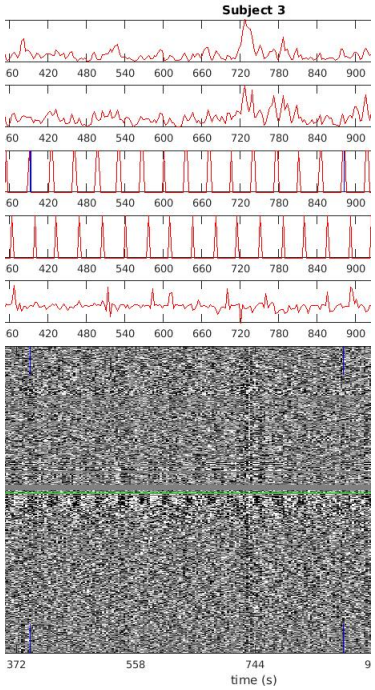


(b) Session 2

FIGURE 5.7 – Diagramme de Power pour le sujet n^o 4.
Le premier graphique correspond à l'évolution de $TAD'(t)$ en mm. Le deuxième graphique correspond à l'évolution de $RAD'(t)$ en deg. La troisième courbe représente la phase de réception de la boisson du paradigme présenté en figure 5.1. La quatrième courbe représente la phase de déglutition du paradigme présenté en figure 5.1. Le cinquième graphique représente le signal de pression mesuré sous l'épiglotte. La carte de gris correspond à la variation de signal par rapport à la ligne de base mesuré dans les voxels de la substance grise et de la substance blanche. La ligne verte marque la séparation entre la substance grise (en haut) et la substance blanche (en bas).

grise qui surviendraient pendant une période du paradigme (réception de la boisson ou déglutition). Sur les figures 5.8 et 5.9, le trait bleu plein identifie l'acquisition correspondant à la perturbation au niveau de l'onset et les deux traits bleus l'identifient au niveau de la variation du signal. Ces figures montrent que certaines perturbations sont dues à la réception de la boisson ou à la déglutition. Cependant, d'autres variations ne sont pas imputables aux paradigmes.

La table 5.3 montre le coefficient de corrélation calculé entre la variation de signal dans la substance blanche et le paradigme (défini comme un signal créneau valant 1 lors des périodes de stimulation et 0 ailleurs). Nous constatons qu'il n'y a presque aucune corrélation existant entre les variations dans la substance blanche et le paradigme à l'exception notable du sujet 2 pour lequel il existe une corrélation négative entre le paradigme de déglutition et les variations du signal dans la substance blanche. Cette table confirme les résultats observés sur les diagrammes de Power.



(a) Sujet n^o 3 - Session 1

FIGURE 5.8 – Variations survenant pendant une période de réception de la boisson.

Sujet n^o	Session 1	Session 2
1	0,03	-0,07
2	0,13	0,20
3	0,12	0,04
4	0,19	0,08

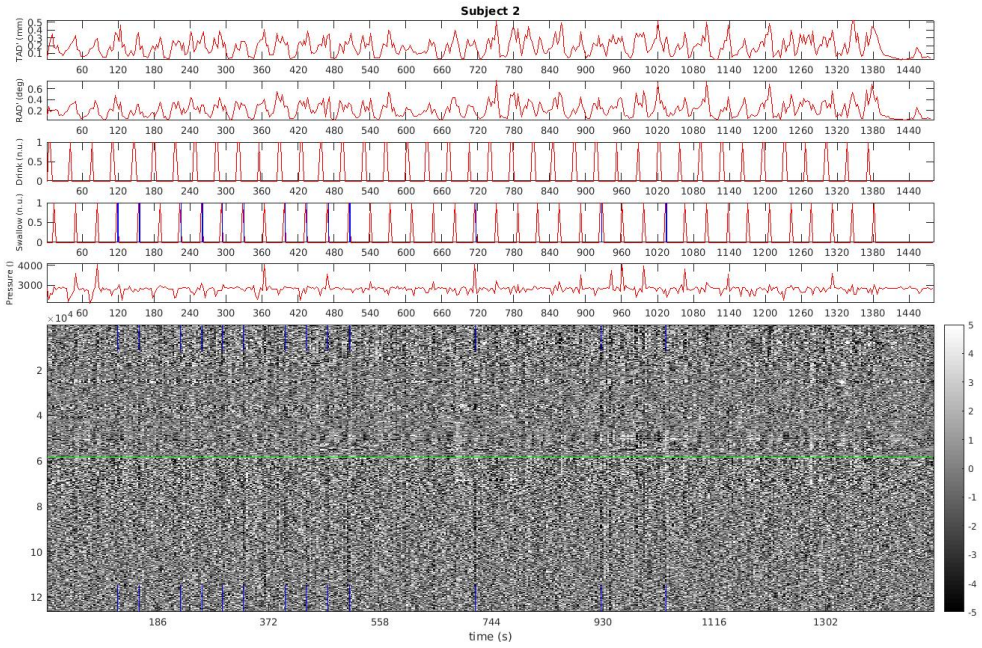
(a) Corrélation entre s_{WM} et l'onset *Drink*

Sujet n^o	Session 1	Session 2
1	0,00	0,04
2	-0,53	-0,56
3	-0,01	-0,07
4	-0,07	0,04

(b) Corrélation entre s_{WM} et l'onset *Swallow*

TABLE 5.3 – Corrélation entre s_{WM} et le paradigme .

Certaines variations dues au mouvement sont synchrones à certaines phases du paradigme, en particulier la phase de déglutition et à moindre échelle à la phase de réception de la boisson. Cela étant, les faibles corrélations observées suggèrent que les artefacts de mouvement ne sont pas synchrones à ces deux phases particulières.



(a) Sujet n^o 2 - Session 1

FIGURE 5.9 – Variations survenant pendant une période de déglutition.

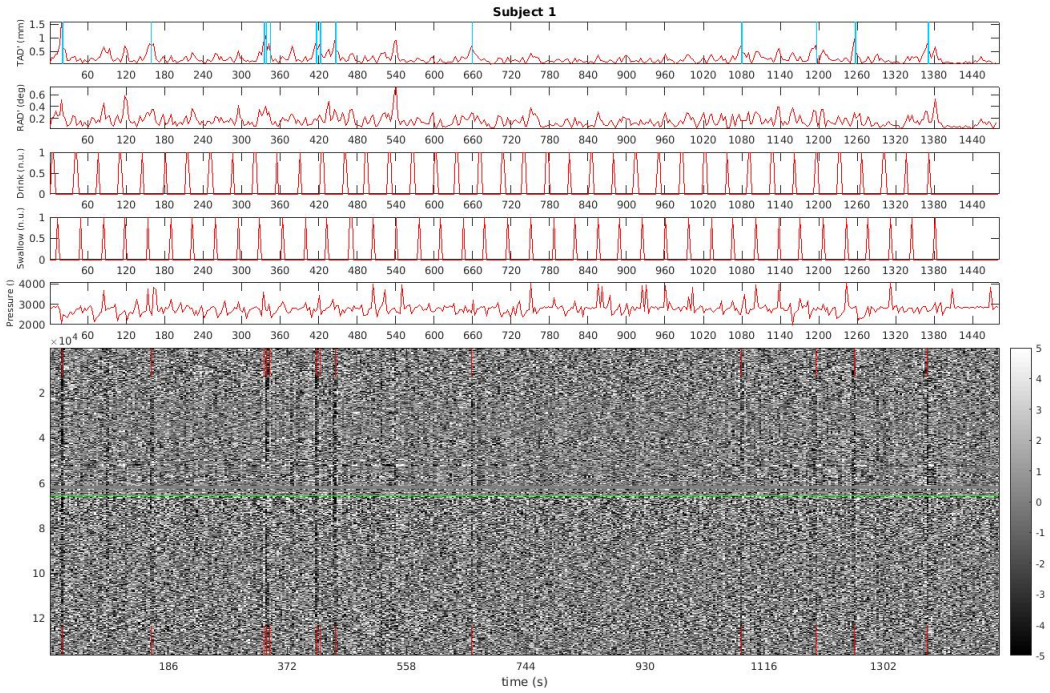
Pour corriger l'impact du mouvement, il n'est donc pas pertinent de masquer les signaux acquis pendant ces phases du paradigme. La partie suivante vise à préciser l'importance des artéfacts se produisant en dehors de ces périodes bien identifiées, mouvements qualifiés donc de sporadiques.

5.3.5 Ces variations sont-elles dues aux mouvements sporadiques ?

Afin de déterminer si les variations observées sont dues aux mouvements sporadiques, on ne considère comme mouvements sporadiques que les valeurs de RAD' et TAD' appartenant au $95^{ème}$ centile (autrement dit les 5% de valeurs les plus élevée).Les figures 5.10 et 5.11 montrent quelques exemples de variations du signal imputables aux mouvements sporadiques. L'acquisition est identifiée sur le graphique TAD'/RAD' par un trait bleu clair et est reporté sur la carte en nuance de gris par des traits rouges. Pour les sujets n^{os} 1, 3 et 4, plusieurs variations de signal surviennent à même temps que des forts mouvements de TAD' ou RAD' (voire les deux en même temps). Ce n'est pas le cas pour le sujet n^o 2 ; toutefois pour ce sujet, une grande partie de ses variations sont imputables aux paradigmes.

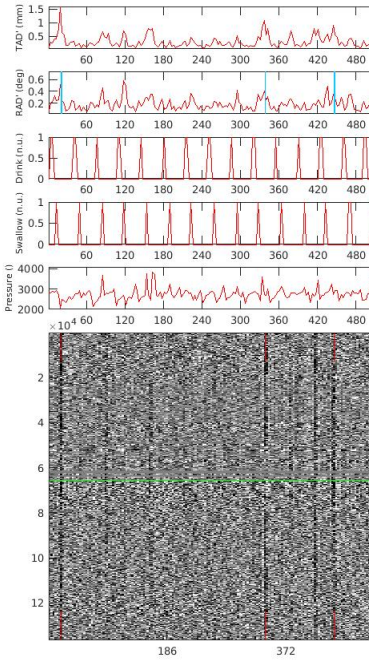
Il existe donc un lien entre les fortes variations de signal observées dans la substance blanche et dans la substance grise et les mouvements sporadiques. Ce lien est confirmé par le coefficient de corrélation entre le signal $s_{WM}(t)$ dans la substance blanche et les paramètres de mouvement montré en Table 5.4. Une corrélation moyenne existe entre TAD' et $s_{WM}(t)$ pour les sujets n^{os} 1, 3 et 4, comme le confirme les figures 5.10 et 5.11. Cependant, il est impossible d'établir un lien entre un paramètre de mouvements particulier et les variations dans la substance grise. En effet, nous observons une variabilité à la fois inter-session et inter-individuelle de la corrélation. Nous pouvons cependant noter que pour les sujets n^{os} 1, 3 et 4, la corrélation est surtout présente avec les dérivées des paramètres de mouvement.

Une grande partie des variations observées dans la substance blanche se produit en même temps qu'un mouvement sporadique. Bien que ce lien soit confirmé par



(a) Sujet n^o 1 - Session 1

FIGURE 5.10 – Variations survenant pendant un fort mouvement sporadique de translation .



(a) Sujet n^o 1 - Session 1

FIGURE 5.11 – Variations survenant pendant un fort mouvement sporadique de rotation .

Sujet n^o	1	2	3	4
$\Delta x(t)$	-0,06	-0,05	-0,12	0,18
$\Delta y(t)$	0,04	0,05	0,06	**0,46**
$\Delta z(t)$	-0,18	0,11	-0,11	-0,32
$roll(t)$	0,02	-0,07	-0,05	**0,44**
$pitch(t)$	-0,02	0,00	0,03	0,02
$yaw(t)$	-0,16	-0,02	-0,02	0,14
$TAD(t)$	-0,11	0,00	-0,17	-0,31
$RAD(t)$	-0,04	0,01	-0,05	-0,37
$FD(t)$	**-0,44**	-0,11	**-0,50**	**-0,49**
$x'(t)$	**-0,46**	0,06	-0,09	0,14
$y'(t)$	0,18	0,22	0,19	**0,41**
$z'(t)$	**-0,53**	0,31	-0,02	**-0,54**
$roll'(t)$	0,13	-0,23	-0,13	0,38
$pitch'(t)$	-0,23	0,22	0,07	0,06
$yaw'(t)$	-0,26	0,31	-0,12	0,34
$TAD'(t)$	**-0,44**	-0,11	**-0,50**	**-0,49**
$RAD'(t)$	-0,21	-0,10	-0,36	**-0,48**

(a) Session 1

Sujet n^o	1	2	3	4
$\Delta x(t)$	0,01	0,09	-0,12	0,06
$\Delta y(t)$	-0,01	0,12	0,03	0,23
$\Delta z(t)$	-0,06	0,05	-0,19	-0,35
$roll(t)$	-0,02	-0,10	0,00	0,31
$pitch(t)$	0,02	0,03	0,07	0,01
$yaw(t)$	0,02	-0,02	0,00	0,20
$TAD(t)$	0,03	-0,04	-0,14	-0,31
$RAD(t)$	0,00	0,06	-0,07	-0,26
$FD(t)$	-0,14	-0,09	**-0,63**	-0,26
$x'(t)$	0,12	-0,04	-0,15	0,05
$y'(t)$	-0,08	0,32	-0,00	0,18
$z'(t)$	-0,24	0,35	**-0,42**	-0,13
$roll'(t)$	-0,10	-0,34	0,04	0,24
$pitch'(t)$	0,13	-0,17	0,14	-0,01
$yaw'(t)$	0,09	-0,18	0,07	0,18
$TAD'(t)$	-0,14	-0,09	**-0,63**	-0,26
$RAD'(t)$	-0,17	-0,08	**-0,46**	-0,29

(b) Session 2

TABLE 5.4 – Corrélation entre $s_{WM}(t)$ et les paramètres de mouvement

une corrélation moyenne, la correction des artéfacts de mouvements par les dérivées des paramètres de mouvement ne nous paraît pas pertinent pour deux raisons. La première raison est que cette relation entre variation dans la substance blanche et mouvement sporadique n'existe que pour certains paramètres du mouvement propre à chaque session. La deuxième raison est qu'en identifiant les variations à l'aide des paramètres de mouvement, les variations liées à l'exécution du paradigme risquent de ne pas être identifiées.

5.3.6 Correction des artéfacts de mouvements

L'analyse précédente et les diagrammes de Power en particulier ont montré que pour prendre en compte le mouvement, qui présente de nombreux événements sporadiques, mieux vaut se fonder sur ses conséquences que sur ses causes. C'est pourquoi la stratégie développée a consisté à identifier les acquisitions présentant des variations atypiques du signal dans la substance blanche et dans la substance grise. Une censure de ces acquisitions a été ensuite utilisée car elle s'est avérée efficace pour des paradigmes présentant un haut degré de mouvement [186].

5.3.6.1 Méthode de l'ellipse de confiance

Pour éliminer les perturbations dues aux mouvements de déglutition ou sporadiques, nous avons choisi d'éliminer les points temporels qui correspondent à des fortes variations dans la substance blanche se répercutant dans la substance grise. Pour cela, nous proposons de tracer la variation de la substance grise en fonction de la variation dans la substance blanche, variation relative par rapport à la ligne de base moyennée sur la session. La figure 5.12 représente le nuage des acquisitions pour la première session du sujet 1. Les points d'une session se répartissent selon une ellipse. De plus, ce diagramme permet de détecter certains points aberrants, dont l'amplitude est très atypique par rapport à l'ensemble de la population. Or ces points correspondent aux traînées observables sur les diagrammes de Power.

Paramètres de l'ellipse de confiance L'idée est d'utiliser le nuage de points pour déterminer les points atypiques. Nous choisissons d'utiliser une ellipse de confiance et d'exclure les points

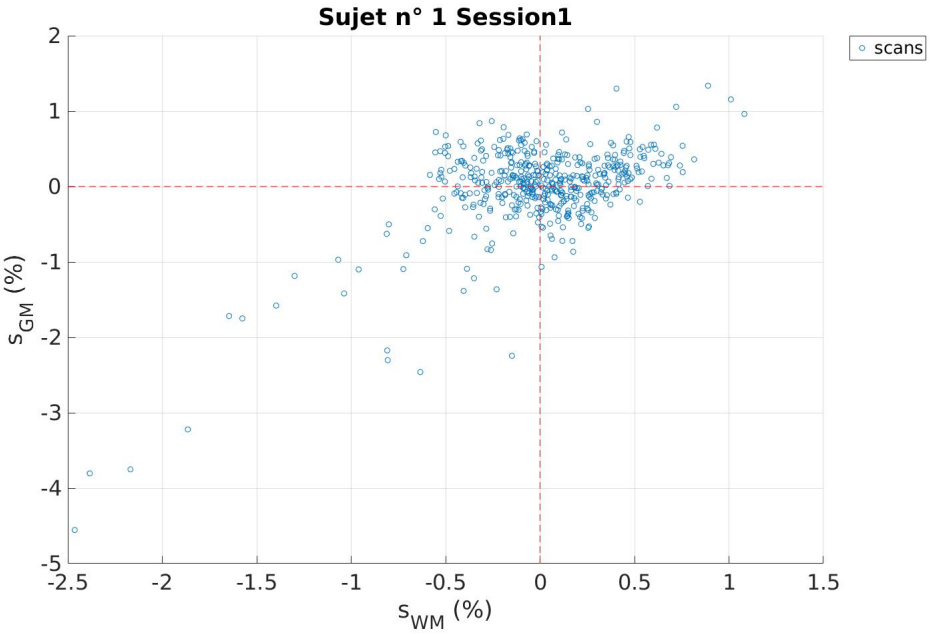


FIGURE 5.12 – Nuages de acquisitions de la session 1 du sujet n^o 1.
Chaque point représente une acquisition. La variation de signal moyennée dans la substance blanche est reportée sur l'axe des abscisses. La variation de signal moyennée dans la substance grise est donéne en ordonnée.

(et donc les acquisitions correspondantes) qui se trouvent à l'extérieur de l'ellipse de confiance [109].

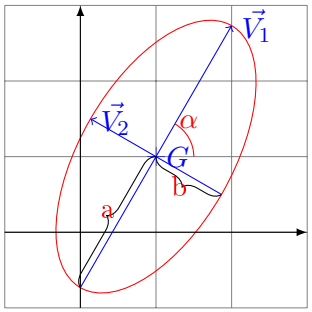


FIGURE 5.13 – Paramètres de l'ellipse.

Le centre de l'ellipse de confiance correspond au centre de gravité du nuage de points. L'orientation de l'ellipse (son angle α) est calculée à l'aide de la matrice de covariance C du nuage de points. Pour cela, on obtient la décomposition en valeurs propres de la matrice C, soit V l'ensemble de ses vecteurs propres et λ l'ensemble de ses valeurs propres. Ces valeurs propres représentent la variance des données dans la direction des vecteurs propres. Notons $\lambda_1 = max(\Lambda)$ la plus grande valeur propre et V_1 le vecteur propre associé, l'angle α est alors défini par $\alpha = \arctan \frac{V_1(y)}{V_1(x)}$ en radians. En prenant comme repère les vecteurs propres de la matrice de covariance ayant pour origine le centre de gravité du nuage de points G, l'équation de l'ellipse s'écrit :

$$\frac{x^2}{\lambda_1} + \frac{y^2}{\lambda_2} = s \tag{5.1}$$

où $s \geq 0$ définit l'échelle de l'ellipse de confiance. Le demi-grand axe a de l'ellipse s'obtient par $a = \sqrt{(s\lambda_1)}$ et le demi-petit axe b s'obtient par $b = \sqrt{(s\lambda_2)}$. Il suffit ensuite de choisir s pour

représenter un intervalle de confiance choisi. Pour calculer s, on fait l'hypothèse que nos données sont échantillonnées selon une loi gaussienne. Le terme gauche de l'équation (5.1) correspond alors à la somme des carrés de deux variables indépendantes normalement distribuées. Or cette somme est distribuées selon la loi du χ_2, à deux degrés de liberté dans notre cas (puisqu'on a deux inconnues). Pour un intervalle de confiance de probabilité p, on peut définir s par $s = Inv - \chi_2(p)$. Une ellipse de confiance avec une probabilité p=0.95 signifie qu'un nouveau couple (x, y) de points aura une probabilité de 95% d'être dans l'ellipse de confiance.

Test d'appartenance à l'ellipse de confiance Pour tester si les couples $(s_{WM}(t), s_{GM}(t))$ appartiennent à l'ellipse de confiance, une solution serait de vérifier l'équation de l'ellipse donnée en (5.1). Cependant, on utilise la distance de Mahalanobis. L'équation de l'ellipse donnée en (5.1) est en effet valable dans le repère défini par les vecteurs propres $V1$ et $V2$. Pour l'utiliser, il faut donc faire un changement de repère : du repère cartésien classique vers le repère de l'ellipse. Notons $\mathcal{E}$ l'ellipse de confiance, la distance de Mahalanobis entre le point $\boldsymbol{p} = (s_{WM}(t), s_{GM}(t))$ et l'ellipse $\mathcal{E}$ est donnée par

$$d(\boldsymbol{p}, \mathcal{E}) = \sqrt{\boldsymbol{p}^\top \boldsymbol{C}^{-1} \boldsymbol{p}} \ .$$

Notons également $\boldsymbol{C}$ la matrice de covariance du nuage de points. la distance de Mahalanobis est simple à calculer et n'implique pas de changement de repère. Pour tester si un point $\boldsymbol{p} = (s_{WM}(t), s_{GM}(t))$ est à l'intérieur de l'ellipse, il suffit de tester si $d(\boldsymbol{p}, \mathcal{E}) \leq \sqrt{s}$ où s est l'échelle choisie (rappelons que $s = Inv - \chi_2(p)$).

Reste alors à choisir la probabilité p de l'intervalle de confiance. Il faut choisir la probabilité p assez grande pour éliminer les perturbations (les traînées noires observables sur les diagrammes de Power) tout en veillant à ne pas éliminer trop de points. La figure 5.14 montre que certains points sont considérés comme aberrants quelle que soit la probabilité $p \in \{0,90; \ldots; 0,99\}$. Pour les autres points à l'inverse, le choix de la probabilité aura toute son importance.

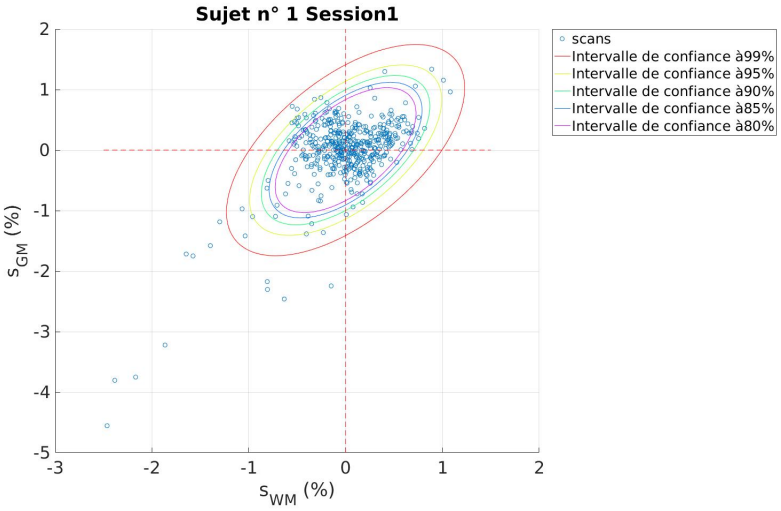
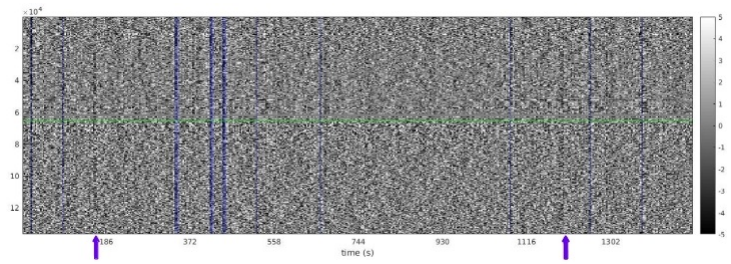


FIGURE 5.14 – Nuage des acquisitions pour la session 1 du sujet n^o 1 et ellipses de confiance à différentes probabilités.

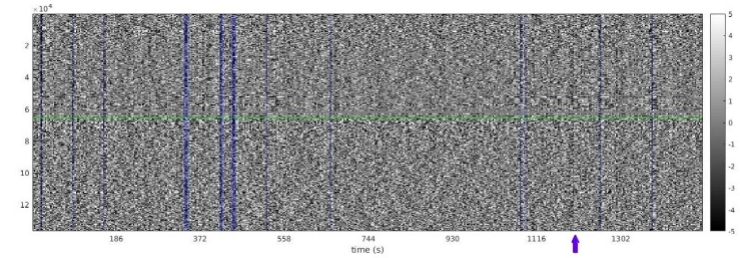
5.3.6.2 Choix de la probabilité pour l'intervalle de confiance

Pour choisir la probabilité, il faut donc s'assurer que toutes les traînées observables sur le diagramme de Power sont considérées comme des perturbations et sont en dehors de l'ellipse de confiance.

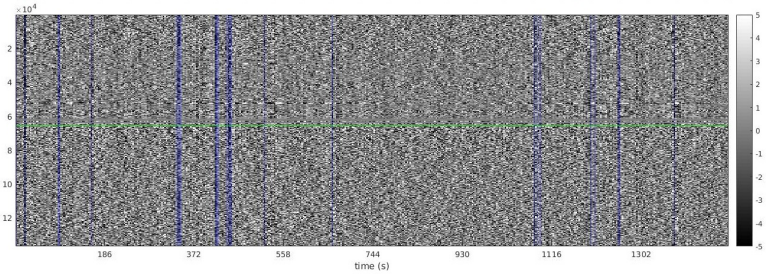
La figure 5.15 montre par exemple que pour le sujet n^o 1, avec la probabilité $p = 0,99$ l'ellipse de confiance englobe des points correspondant à des traînées noires dans le diagramme de Power



(a) Diagramme de Power avec une ellipse de confiance à 99%



(b) Diagramme de Power avec une ellipse de confiance à 97%



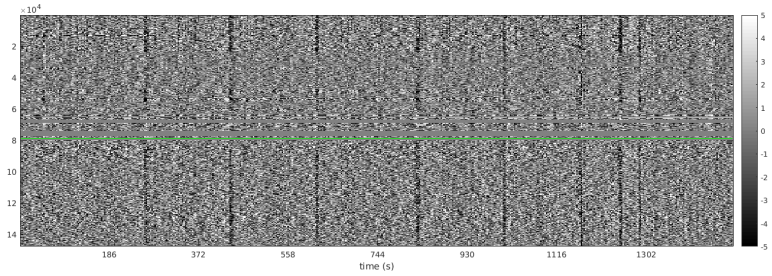
(c) Diagramme de Power avec une ellipse de confiance à 95%

FIGURE 5.15 – Évolution du diagramme de Power pour la session 1 du sujet n^o 1. Les lignes bleues représentent les points atypiques.

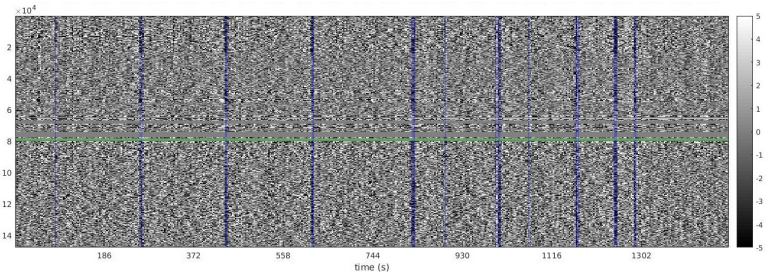
(flèches violettes sur la figure 5.15(a)). Avec la probabilité $p = 0.97$, toutes les fortes variations de signal (flèches violettes sur la figure 5.15(b)) ne sont pas situées en dehors de l'ellipse et ne sont donc pas considérées comme aberrantes. À partir de la probabilité $p = 0,95$, toutes les traînées noires sont considérées comme des perturbations.

Sur la base de l'analyse des 4 sujets, un seuil de $p = 0,95$ parait correct. La figure 5.16 montre l'effet de la correction pour la session 1 du sujet n^o 4 (qui présente des variations sur un intervalle temporel plutôt que ponctuellement). Les figures des autres sessions et sujets sont disponibles en Annexe E.1. Le seuil de 95% permet d'éliminer tous les points atypiques correspondant aux traînées identifiées à l'aide des diagrammes de Power, et mêmes des points non identifiés comme aberrant par les diagrammes de Power.

Nous disposons ainsi d'une méthode pour identifier des points temporels aberrants définis comme des points temporels où la variation de signal dans la substance blanche est élevée et se répercute dans la substance grise. Cette méthode est indépendante des paramètres de réalignement rigide et des paramètres de mouvement dérivés des paramètres de réalignement. Elle ne dépend que des données acquises. Elle s'appuie sur le tracé de l'ellipse de confiance dans le nuage de points représentant la variation de signal dans la substance blanche en fonction de la variation du signal dans la substance grise. Le seul paramètre de la méthode est le choix de l'intervalle de confiance.



(a) Original



(b) Après censure

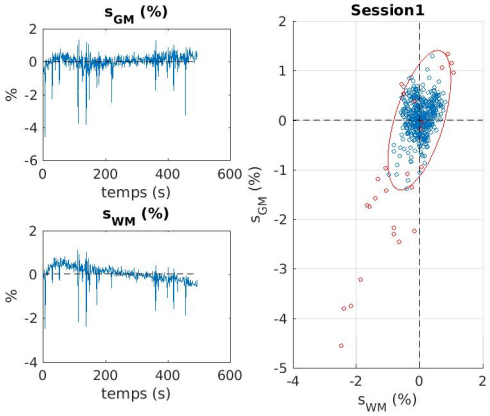
FIGURE 5.16 – Diagramme de Power avant/après censure pour la session 1 du sujet n^o 4.

5.4 Choix du filtre de passe-haut

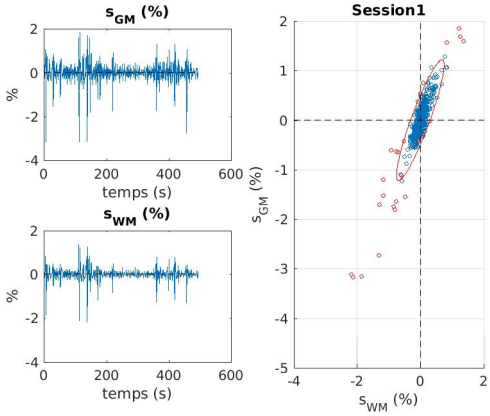
Un filtre passe-haut sur les données normalisées est souvent appliqué comme prétraitement sur les données d'IRMf avec une période de coupure de 128 secondes. L'idée est d'enlever les variations lentes, issues des dérives du système d'imagerie. Notre but étant de parcelliser les signaux les plus similaires, nous avons étudié l'influence de la période de coupure sur les variations du signal. Nous voulons en effet regrouper les voxels qui répondent de manière similaire au paradigme et non les voxels qui présentent les mêmes artéfacts de signal. De plus, notre méthode de censure est fondée sur la variation de signal dans la substance grise et dans la substance blanche calculée à partir de la ligne de base. Or, l'application d'un filtre passe-haut modifie la ligne de base. Il faut donc choisir une période de coupure suffisante pour corriger la dérive temporelle sans introduire de nouveaux artéfacts. Ces variations du signal ont été moyennées sur un masque de substance grise en ne gardant que les voxels dont l'appartenance à la substance grise est supérieure à 0,9 et sur un masque de substance blanche en ne gardant que les voxels dont l'appartenance à la substance blanche est supérieure à 0,9 (**cf. Section A.1**). Les points déterminés comme aberrants par correction du mouvement, introduite en section 5.3.6, pour les données filtrées avec une période de coupure de 128 s, sont représentés en rouge dans les nuages de points afin de voir leur évolution suivant les périodes de coupure choisies.

Les figures 5.17 et 5.18 montrent l'évolution des variations de signal et le nuage de points utilisé pour tracer l'ellipse de confiance pour deux sujets en fonction de la période de coupure choisie. Quelle que soit la période de coupure choisie, l'ellipse est inclinée traduisant une relation entre la variation de signal dans la substance grise et la variation de signal dans la substance blanche. Lorsqu'aucun filtre passe-haut n'est appliqué, la dérive temporelle peut être observée. Dès qu'un filtre passe-haut est appliqué, la dérive temporelle est corrigée. Cependant, lorsque le filtre passe-haut est choisi avec une période de coupure de 512 s, des oscillations de la ligne de base sont observables. Nous remarquons également que l'application d'un filtre passe-haut accentue les variations de signal négatives. Concernant les points aberrants détectés, il est notable que leur position évolue avec l'augmentation de la fréquence de coupure. L'augmentation de la période de coupure entraîne une accentuation des variations négatives ce qui se traduit par un décalage

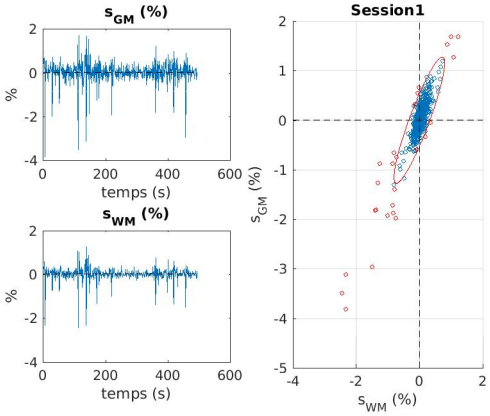
de l'ellipse vers le quadrant en bas à gauche. De plus, les points ayant une variation positive à la fois dans la substance grise et dans la substance blanche se rapprochent de l'ellipse de confiance (jusqu'à y entrer avec une période de coupure de 512 s voire 256 s).



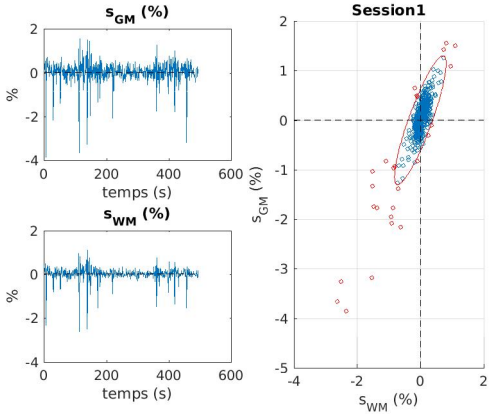
(a) Sans filtre



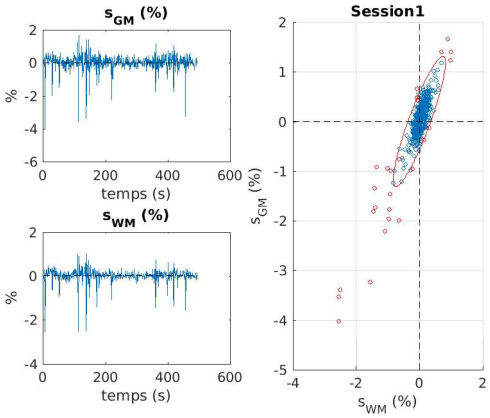
(b) Passe-haut avec une cop=32 s



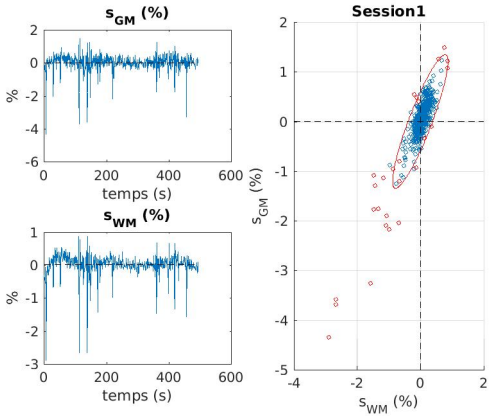
(c) Passe-haut avec une cop=64 s



(d) Passe-haut avec une cop=128 s



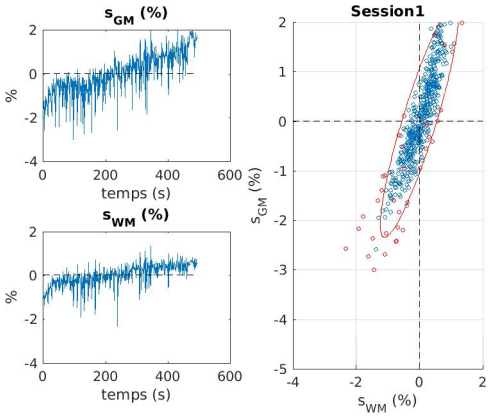
(e) Passe-haut avec une cop=256 s



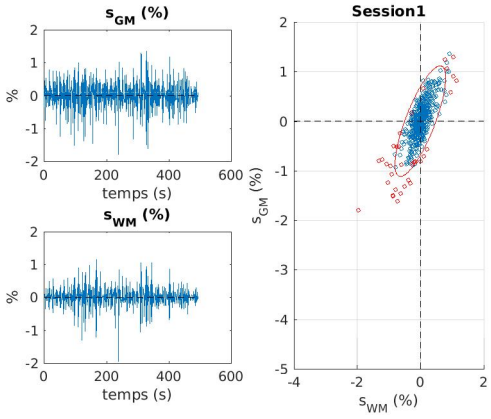
(f) Passe-haut avec une cop=512 s

FIGURE 5.17 – Sujet n^o 1 Session ^{o} 1 : Évolution des variations du signal en fonction de la période de coupure (cop).

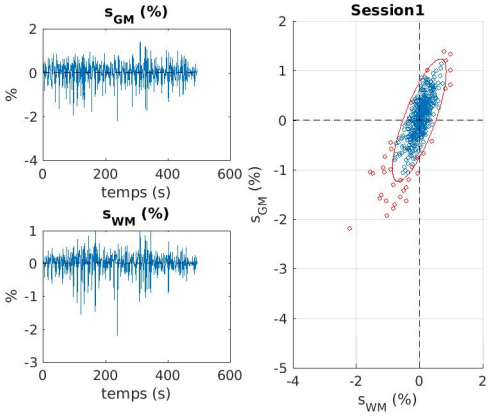
Pour corriger la dérive temporelle présente dans les données, un filtrage temporel est nécessaire. Cependant, le choix de la période de coupure influence la détermination des points aberrants à l'aide de la méthode de l'ellipse de confiance en modifiant la moyenne temporelle du



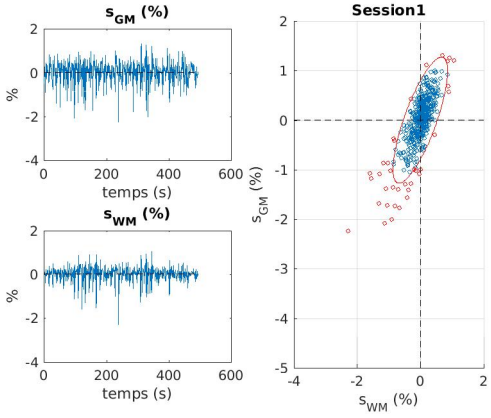
(a) Sans filtre



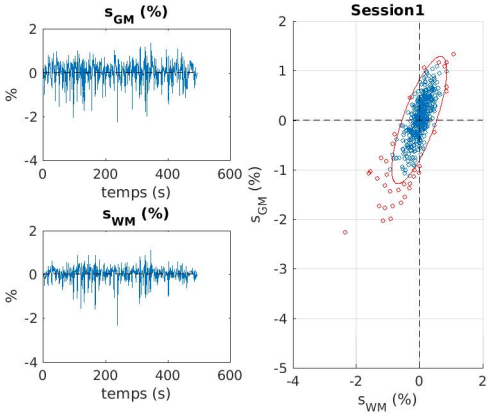
(b) Passe-haut avec une cop=32 s



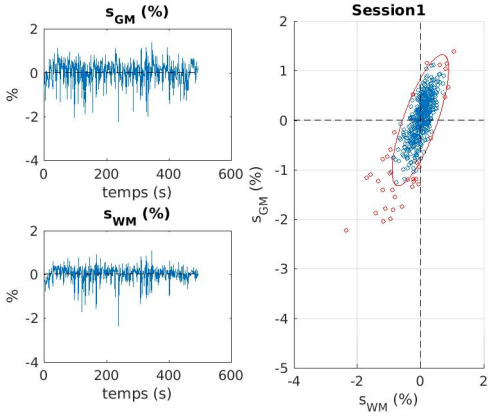
(c) Passe-haut avec une cop=64 s



(d) Passe-haut avec une cop=128 s



(e) Passe-haut avec une cop=256 s



(f) Passe-haut avec une cop=512 s

FIGURE 5.18 – Sujet n^o 2 Session ^{o} 1 : Évolution des variations du signal en fonction de la période de coupure (cop).

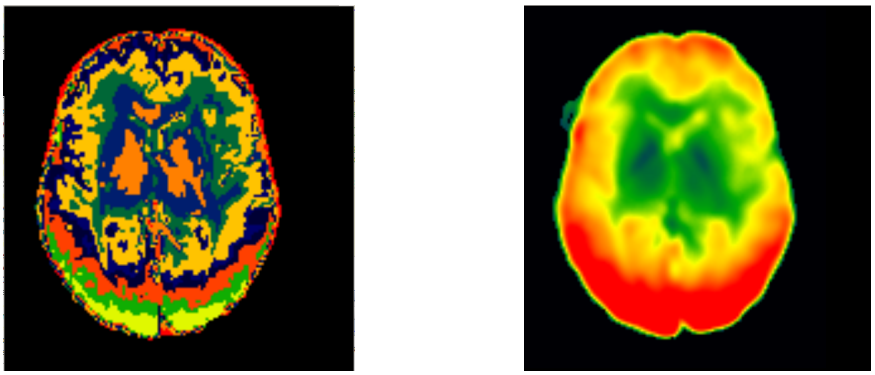


FIGURE 5.19 – Comparaison de la parcellisation sur le signal (à gauche) et du bêta constant issu du GLM (à droite).

signal. Si une période de coupure de 32 secondes permet de corriger la dérive, une période de coupure trop élevée (512 secondes pour nos données) peut faire apparaître des oscillations dans la variation de signal. Il est également à noter que lorsque la période de coupure augmente, les pics de variations négatifs s'accentuent alors que les pics de variations positives tendent à diminuer. **La période de coupure par défaut de 128 secondes corrige la dérive temporelle sans introduire d'oscillations. De plus, par rapport aux points aberrants, l'ellipse de confiance apparaît stable entre une période de coupure de 128 s et une période de coupure de 256 s. C'est pourquoi un filtre passe-haut avec une période de coupure de 128 secondes a été appliqué à nos données.**

5.5 Quel signal parcelliser ?

Le signal d'IRMf étant prétraité et débarrassé autant que possible des perturbations liées au mouvement, il reste à définir l'information utilisée par l'algorithme agglomératif.

La variation relative du signal est utilisée pour s'affranchir de la ligne de base. En effet, cette ligne varie dans le cerveau en fonction de facteurs instrumentaux (hétérogénéités du champ statique et des réponses des antennes) et non neuronaux. Cette variation est beaucoup plus importante (>10%) que le contraste BOLD (1%). Elle détermine donc le résultat de la parcellisation si la ligne de base n'est pas soustraite. La variation de signal (percent signal change (PSC)) est définie par :

$$PSC(v,t) = \frac{s(v,t) - \overline{s(v)}}{\overline{s(v)}}$$

où $s(v,t)$ représente le signal mesuré dans un voxel v au temps t et $\overline{s(v)}$ correspond à la moyenne du signal temporel dans le voxel v. La variation de signal est une mesure indirecte de la réponse aux stimuli mesurée dans chaque voxel. La figure 5.19 compare la parcellisation obtenue sur l'ensemble du cerveau lorsqu'on considère le signal après filtrage et censure des points temporels aberrants (figure de gauche) et la carte du bêta constant issu du GLM (figure à droite) pour le même sujet donné. La carte du bêta correspond à la moyenne sur la période de repos. La parcellisation correspond à une répartition en dix parcelles et pour plus de clarté, chaque parcelle est représentée par une couleur (du bleu au rouge en fonction de l'indice de la parcelle). Les couleurs sur la carte du bêta code la valeur de la constante. **La parcellisation sur le signal après prétraitement ressemble donc à la carte du bêta constant, ce qui montre que la parcellisation est dominée par la moyenne du signal et non par la réponse aux stimuli.**

Par la suite, pour limiter le nombre de voxels considérés, on a restreint les voxels à ceux dans la substance grise avec un seuil d'appartenance fixé à 0,5 (**cf. Section A.1**). Cela permet de

réduire le nombre de voxels de moitié.

La parcellisation individuelle consiste à parcelliser le signal obtenu pour chaque sujet séparément obtenant ainsi, pour un nombre de parcelles donné, une parcellisation par sujet. L'avantage est que la parcellisation individuelle reflète l'organisation spatiale de la réponse aux stimuli propre au sujet. En revanche, il n'y a pas de parcellisation commune aux sujets car rien ne garantit que les parcelles vont se recouvrir. À l'inverse, il est possible de réaliser une parcellisation de groupe après l'étape de normalisation qui assure que les images de chaque sujet sont dans le même espace de coordonnées. Nous proposons d'obtenir une parcellisation de groupe en concaténant les signaux de chaque sujet, c'est-à-dire qu'on forme pour chaque voxel le signal :

$$PSC(v) = [PSC_1(v,1), \ldots, PSC_1(v,n_\mathrm{T}), \ldots, PSC_i(v,1), \ldots, PSC_i(v,n_\mathrm{T}), \ldots, PSC_{n_\mathrm{S}}(v,1), \ldots, PSC_{n_\mathrm{S}}(v,n_\mathrm{T})]$$

où $PSC_i(v,t)$ est la variation de signal mesurée dans le voxel v au temps t pour le sujet i. L'avantage d'une parcellisation de groupe est que les sujets partagent les mêmes parcelles, ce qui permet par exemple de calculer la corrélation entre parcelles pour chaque sujet puis de moyenner les corrélations sur l'ensemble des sujets. Cependant, dans le cas de la concaténation, les voxels sont regroupés selon la similarité du signal concaténé ce qui ne garantit pas que le signal est similaire entre les sujets. Mais, la concaténation permet de prendre en compte la censure des points temporels atypiques qui s'effectue à l'échelle individuelle.

Nous avons proposé deux prétraitements inédits pour traiter nos données : une normalisation avec *warping* à l'échelle individuelle qui permet de recouvrir du signal et une méthode de censure des EPIs. Cette méthode de censure est fondée uniquement sur le signal mesuré dans la substance blanche et dans la substance grise.Nous proposons de parcelliser les données à partir des variations du signal après censure des EPIs et d'obtenir une parcellisation de groupe commune à partir de la concaténation des variations du signal censurées des différents sujets. Les résultats de parcellisation sont présentés au chapitre 9. Le chapitre suivant détaille la méthode de parcellisation utilisée.

Chapitre 6

Méthode : Algorithme de Ward

Ce chapitre présente en détail le clustering hiérarchique agglomératif ainsi que l'algorithme de Ward et ses implémentations séquentielles et parallèles. La première partie est consacrée aux algorithmes hiérarchiques agglomératifs. La deuxième partie détaille l'algorithme de Ward et son implémentation. La troisième section détaille les implémentations parallèles.

6.1 Clustering hiérarchique agglomératif

6.1.1 Principe général

Nous voulons regrouper n_{V} vecteurs ayant n_{T} caractéristiques en n_{C} sous-groupes. Notons ces vecteurs $\boldsymbol{x}_i$ de dimension n_{T}. Appelons $\boldsymbol{X}$ l'ensemble de ces vecteurs (leur concaténation).

Pour obtenir n_{C} sous-groupes à partir de $\boldsymbol{X}$, il faut définir selon quels critères on souhaite regrouper les vecteurs $\boldsymbol{x}_i$ Par exemple, sur la figure 6.1, on dispose de vingt carrés qu'on souhaite regrouper selon leur couleur. Pour cela, on peut représenter la couleur des carrés sous le format RGB où chaque couleur est décomposée en fonction de sa portion de rouge, de sa portion de vert et de sa portion de bleu. On veut maintenant regrouper ces carrés en trois parcelles : une parcelle pour les bleus, une parcelle pour les rouges et une parcelle pour les verts.

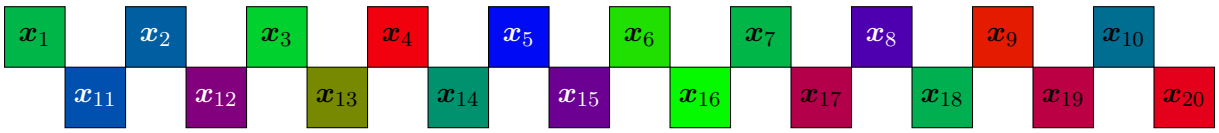


FIGURE 6.1 – Exemple : Regrouper les carrés en fonction de leur couleur.

Dans l'exemple présenté à la figure 6.1, on a donc $n_{\mathrm{T}} = 3$ et $n_{\mathrm{V}} = 20$. La matrice $\boldsymbol{X}$ est donnée par :

$$\boldsymbol{X} = \begin{pmatrix} 0.00 & 0.00 & 0.00 & 1.00 & 0.00 & 0.14 & 0.00 & 0.44 & 1.00 & 0.00 & 0.00 & 1.00 & 0.86 & 0.00 & 0.74 & 0.02 & 1.00 & 0.00 & 1.00 & 1.00 \\ 1.00 & 0.58 & 1.00 & 0.00 & 0.04 & 1.00 & 1.00 & 0.00 & 0.12 & 0.76 & 0.46 & 0.00 & 1.00 & 1.00 & 0.00 & 1.00 & 0.00 & 1.00 & 0.00 & 0.00 \\ 0.40 & 1.00 & 0.22 & 0.06 & 1.00 & 0.00 & 0.40 & 1.00 & 0.00 & 1.00 & 1.00 & 0.96 & 0.00 & 0.76 & 1.00 & 0.00 & 0.42 & 0.46 & 0.36 & 0.12 \end{pmatrix}$$

Revenons maintenant à la méthode. Les vecteurs seront donc regroupés en fonction de leur similarité. La définition de la similarité dépend de ce que contiennent les vecteurs. Cette similarité est calculée en prenant en compte les n_{T} dimensions des vecteurs $(\boldsymbol{x}_i)_{i\in\{1,\dots,n_{\mathrm{V}}\}}$. Plutôt que de parler de similarité, on définit une distance entre les vecteurs : plus la distance entre deux vecteurs est petite, plus les vecteurs sont proches. Plusieurs métriques sont possibles pour calculer la distance entre les observations. On peut utiliser la distance euclidienne, le coefficient de corrélation de Pearson ρ comme similarité et définir une distance par $1 - \rho$, la distance de Manhattan,...

Une fois que la matrice de distances (ou de similarités) est calculée entre les n_{V} vecteurs, on peut commencer à regrouper les vecteurs. Cependant, lorsqu'on regroupe les vecteurs, il faut

mettre à jour la matrice de distances en calculant la distance entre les nouveaux sous-groupes et les vecteurs restants, et entre les sous-groupes. Concrètement, reprenons l'exemple des carrés de couleur et supposons qu'à la première étape, on regroupe deux carrés rouges, il nous faut déterminer comment représenter ces deux carrés rouges par rapport aux autres carrés afin de calculer la distance entre les deux carrés rouges et les carrés restants. La question qui se pose dès qu'on a regroupé plusieurs éléments en sous-groupes est donc comment calculer la distance entre deux sous-groupes (éventuellement unitaires). Cette distance entre sous-groupes est le critère d'agglomération. Il existe plusieurs critères d'agglomération qui seront détaillés plus bas (**cf. Section 6.1.2**).

Supposons que nous disposons d'une distance et d'un critère d'agglomération pour regrouper les vecteurs. Qu'appelle-t-on un algorithme de clustering hiérarchique agglomératif ? Un tel algorithme commence par mettre chaque élément dans une parcelle (un groupe). Autrement dit, au commencement, chaque parcelle contient un unique élément et est donc de taille unitaire. Ensuite, on cherche les deux parcelles les plus similaires en calculant le critère d'agglomération pour chaque paire de parcelles. On regroupe ces deux parcelles en une seule parcelle. Puis on calcule la distance entre cette nouvelle parcelle et les parcelles restantes. L'idée est d'itérer le processus de groupement de deux parcelles deux à deux : à chaque étape, on regroupe les deux parcelles les plus proches et on les fusionne en une seule parcelle. L'algorithme se termine lorsqu'il ne reste plus qu'une seule parcelle. Autrement dit, le regroupement est terminé lorsque tous les éléments sont dans une même parcelle.

La figure 6.2 illustre le principe du clustering hiérarchique agglomératif. Sur la figure, les ovales correspondent à des parcelles. La notation C_j^k indique la j^{ème} parcelle à l'itération k. À la première itération, chaque parcelle n'est composée que d'un seul élément : on a donc n_V parcelles. À chaque itération, on regroupe deux parcelles en une seule : le nombre de parcelles diminue donc de un à chaque itération. Il faut ainsi $n_\mathrm{V} - 1$ itérations pour obtenir une parcelle contenant tous les éléments.

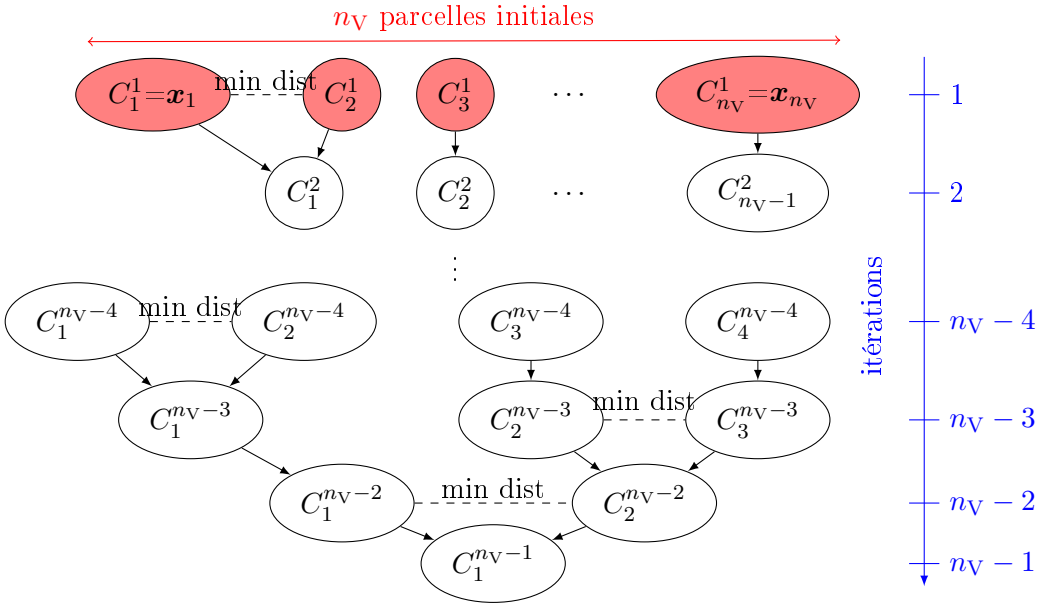


FIGURE 6.2 – Clustering hiérarchique agglomératif.

Comme le montre la figure 6.2, l'algorithme aboutit au regroupement des vecteurs en une seule parcelle, ce qui n'est pas très intéressant. Cependant, chaque ligne de la figure 6.2 représente une parcellisation à une échelle donnée c'est-à-dire pour un nombre de parcelles donné. Soit n_C le nombre de parcelles souhaité, la parcellisation à l'échelle n_C s'obtient à l'itération $n_\mathrm{V} - n_\mathrm{C}$. Pour une partition en n_C parcelles, l'algorithme pourrait se terminer à l'itération $n_\mathrm{V} - n_\mathrm{C}$. Il

existe plusieurs raisons pour poursuivre l'algorithme jusqu'à l'obtention d'une seule parcelle. La première raison est qu'en général, on ne connaît pas *a priori* le nombre de parcelles optimal et dérouler l'algorithme jusqu'au bout permet d'obtenir l'ensemble des échelles de parcellisation possibles. Les critères pour déterminer le nombre optimal de parcelles sont détaillés en section 6.1.5. La deuxième raison est qu'il est plus rapide d'effectuer une fois l'algorithme que de le relancer plusieurs fois. Elle sera détaillée dans la section 6.2.

Le qualificatif d'agglomératif vient du fait qu'on part des plus petites parcelles possibles pour faire grossir au fur et à mesure des étapes les parcelles. C'est un algorithme de bas en haut. Le clustering est dit hiérarchique car le regroupement des parcelles à une étape donnée dépend des regroupements effectués à toutes les étapes précédentes. De ce fait, l'algorithme est sensible aux données aberrantes.

Le clustering hiérarchique agglomératif est déterministe. On obtiendra le même résultat pour plusieurs exécutions. À l'inverse, l'algorithme des k-moyennes est non déterministe car le résultat dépend du choix des clusters initiaux et donc si on change ces clusters initiaux le résultat peut être différent d'une exécution à l'autre.

6.1.2 Critères d'agglomération

Nous avons vu plus haut que pour regrouper les vecteurs, il faut définir comment calculer la distance entre deux groupes. Sont présentés ici les critères d'agglomérations les plus connus (la liste n'étant pas exhaustive).

Les critères d'agglomération les plus connus sont :

— le critère du saut minimal (*single linkage*) : la proximité entre deux parcelles est donnée par la proximité entre leurs deux plus proches vecteurs. Soit C_i^k et C_i^k deux parcelles, la dissimilarité entre C_i^k et C_j^k est donnée par

$$d(C_i^k, C_j^k) = \min \left\{ \delta(\boldsymbol{x}_i, \boldsymbol{x}_j), \boldsymbol{x}_i \in C_i^k, \boldsymbol{x}_j \in C_j^k \right\} \ ;$$

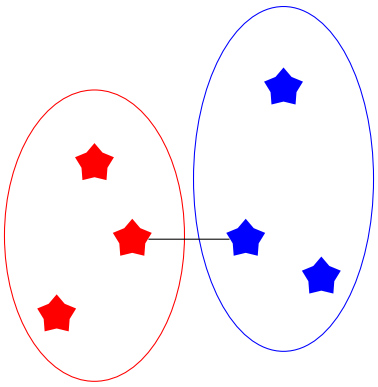


FIGURE 6.3 – Critère du saut minimal.

— le critère du saut maximal (*complete linkage*) : la proximité entre deux parcelles est donnée par la proximité entre les deux vecteurs les plus éloignés. Soit C_i^k et C_i^k deux parcelles, la dissimilarité entre C_i^k et C_j^k est donnée par

$$d(C_i^k, C_j^k) = \max \left\{ \delta(\boldsymbol{x}_i, \boldsymbol{x}_j), \boldsymbol{x}_i \in C_i^k, \boldsymbol{x}_j \in C_j^k \right\} \ ;$$

— le lien moyen (*average linkage* ou *Unweighted pair group method with arithmetic mean*) : la proximité entre deux parcelles est la moyenne arithmétique des proximités sur l'ensemble

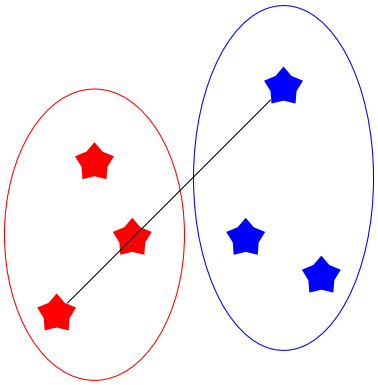


FIGURE 6.4 – Critère du saut maximal.

des paires des vecteurs de la forme (un vecteur d'une parcelle, un vecteur de l'autre parcelle). Soit C_i^k et C_i^k deux parcelles, la dissimilarité entre C_i^k et C_j^k est donnée par

$$d(C_i^k, C_j^k) = \frac{1}{\left|C_i^k\right| \left|C_j^k\right|} \sum_{\boldsymbol{x}_i \in C_i^k} \sum_{\boldsymbol{x}_j \in C_j^k} \delta(\boldsymbol{x}_i, \boldsymbol{x}_j) \ ;$$

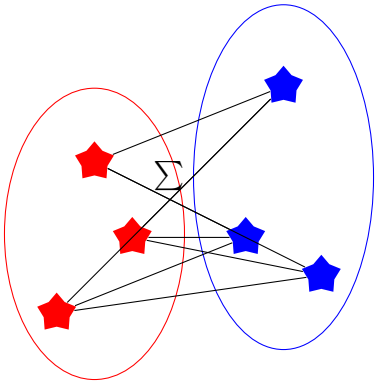


FIGURE 6.5 – Lien moyen.

— le lien moyen équilibré (*Weighted pair group method with arithmetic mean*) qui est une version modifiée du lien moyen : à chaque fusion de parcelles, les deux parcelles participent autant au calcul de la distance avec les autres parcelles quelle que soit leur taille. Soit C_i^k et C_i^k les deux parcelles fusionnées à l'itération k, alors la dissimilarité entre la parcelle $C_{i\cup j}^{k+1}$ et la parcelle C_l^{k+1} est donnée par

$$d(C_{i\cup j}^{k+1}, C_l^{k+1}) = \frac{1}{2}\left(d\left(C_i^k, C_l^k\right) + d\left(C_i^k, C_l^k\right)\right) \ ;$$

— la méthode du centroïde (*centroid linkage* ou *Unweighted centroid clustering*) : la proximité entre deux parcelles est donnée par la distance entre les centroïdes des deux parcelles. Soit C_i^k et C_i^k deux parcelles, la dissimilarité entre C_i^k et C_j^k est donnée par

$$d(C_i^k, C_j^k) = \delta\left(\overline{C_i^k}, \overline{C_j^k}\right) \ ;$$

— la méthode du centroïde équilibré ou méthode de la médiane (*Weighted centroid clustering*) qui est une version modifiée de la méthode du centroïde : la proximité entre deux

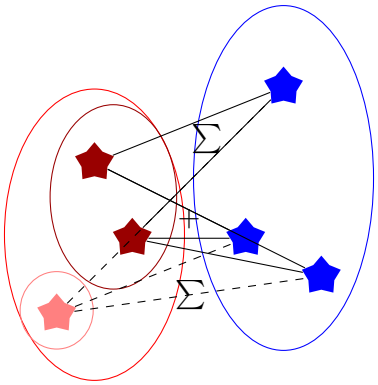


FIGURE 6.6 – Lien moyen équilibré.

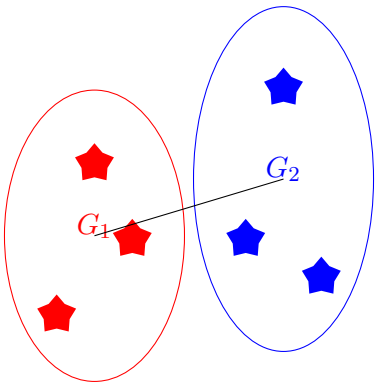


FIGURE 6.7 – Méthode du centroïde.

parcelles est donnée par la distance entre leurs centroïdes géométriques.Soit C_i^k et C_i^k deux parcelles, la dissimilarité entre C_i^k et C_j^k est donnée par

$$d(C_i^k, C_j^k) = \delta\left(\tilde{C_i^k}, \widetilde{C_j^k}\right) \quad ,$$

avec $\widetilde{C_i^k}$ le centroïde géométrique. Si la parcelle C_i^k est née de la fusion des parcelles $C_{i_1}^{k-1}$ et $C_{i_2}^{k-1}$, alors le centroïde géométrique est défini récursivement par :

$$\widetilde{C_i^k} = \frac{1}{2}\left(\widetilde{C_{i_1}^{k-1}} + \widetilde{C_{i_2}^{k-1}}\right) \quad ;$$

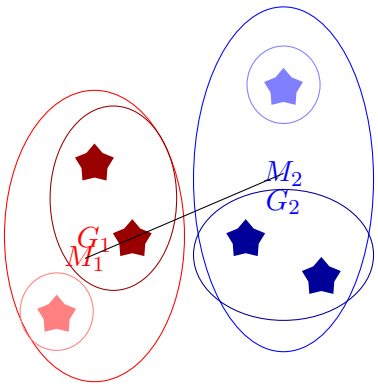


FIGURE 6.8 – Méthode de la médiane.

Sur la figure 6.8, M_1 et M_2 représentent les centroïdes géométriques de chacune des parcelles. Le changement de couleurs représente des parcelles qui ont été fusionnées pour donner les deux parcelles considérées.

— le critère de Ward : on maximise l'augmentation de l'inertie inter-classe qui mesure la dispersion des parcelles par rapport à la moyenne des vecteurs $(\boldsymbol{x}_i)_{i\in\{1,\dots,n_V\}}$. Le critère de Ward correspond également à minimiser l'augmentation de la somme des carrés. Cette augmentation correspond à la somme des carrés de la parcelle jointe (issue de l'union des deux parcelles) à laquelle on soustrait la somme des carrés des deux parcelles. On cherche la paire de parcelles pour laquelle cette différence est minimale. Soit C_i^k et C_i^k deux parcelles, la distance entre C_i^k et C_j^k est donnée par

$$d(C_i^k, C_j^k) = \frac{\left|C_i^k\right|\left|C_j^k\right|}{\left|C_i^k\right| + \left|C_j^k\right|}\delta\left(\overline{C_i^k}, \overline{C_j^k}\right).$$

6.1.2.1 Exemple

Appliquons maintenant les différents critères sur l'exemple des carrés colorés. Les résultats ont été obtenus à l'aide des fonctions linkage et cluster de Matlab. Sur les figures 6.9 à 6.15 chaque ligne de carrés correspond à une parcelle différente et les trois lignes permettent donc de visualiser le découpage des vingt carrés initiaux en trois parcelles.

Regardons le résultat obtenu avec le critère du saut minimal donné à la figure 6.9. Les parcelles sont déséquilibrées en taille. La parcelle sur la ligne du bas regroupe les carrés de couleur rouge. La parcelle sur la ligne du haut comporte un seul carré de couleur vert kaki. Tous les autres carrés ont été regroupés dans la parcelle du milieu formant une longue chaîne mélangeant les couleurs.

Passons maintenant à la parcellisation avec le critère du saut maximal montrée en Figure 6.10. La parcelle du bas regroupe des carrés de couleur bleue avec le dernier carré tirant vers le vert. La parcelle du milieu regroupe des carrés de couleur verte. En revanche, la parcelle du haut regroupe les carrés de couleurs rouge et violet ainsi qu'un carré bleu foncé, que visuellement on ne rangerait pas dans cette parcelle mais dans la parcelle du bas.

La parcellisation obtenue en utilisant le lien moyen est donnée à la figure 6.11. La parcelle du bas regroupe les carrés bleus. La parcelle du milieu regroupe les carrés verts. La parcelle du haut regroupe les carrés rouges et les carrés violets. Cette parcellisation est cohérente visuellement (même si on pourrait préférer regrouper les carrés violets avec les carrés bleus).

Le résultat obtenu avec le lien moyen équilibré est présenté en Figure 6.12. La parcelle du bas regroupe les carrés violets et le carré bleu foncé. La parcelle du milieu regroupe des carrés verts et des carrés bleus. La parcelle du haut regroupe les carrés rouges et le carré vert kaki, ce qui visuellement ne correspond pas.

La méthode du centroïde montrée en Figure 6.13 donne un résultat semblable à celui du lien moyen. La différence entre les deux parcellisations est que pour la méthode du centroïde, la parcelle du bas contient les carrés bleus et un carré vert, qui pour la parcellisation avec le lien moyen est dans la parcelle 2 avec les autres carrés verts.

Regardons maintenant la parcellisation obtenue avec le critère de la médiane. La parcelle du bas regroupe les carrés rouges. La parcelle du milieu regroupe les carrés violet et le carré bleu foncé. La parcelle du haut regroupe des carrés bleus et des carrés verts. La parcelle du bas et la parcelle du milieu sont cohérentes visuellement. Mais pour la parcelle du haut, visuellement, on séparerait les carrés bleus des carrés verts. Cette constatation semble indiquer que le découpage en trois parcelles n'est peut-être pas le plus adapté pour l'exemple choisi et qu'un découpage en quatre parcelles conviendrait éventuellement mieux.

Finissons l'exemple avec la méthode de Ward montré en Figure 6.15. La parcelle du bas regroupe les carrés verts. La parcelle du milieu regroupe des carrés bleus, les carrés violet et un

carré vert. La parcelle du haut regroupe les carrés rouges. Excepté le carré vert dans la parcelle du milieu, le découpage est visuellement correct.

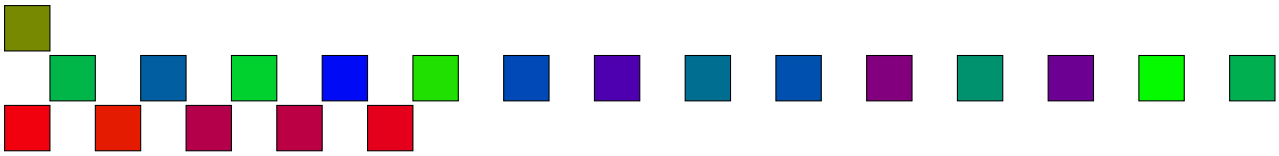


FIGURE 6.9 – Exemple : Critère du saut minimal.

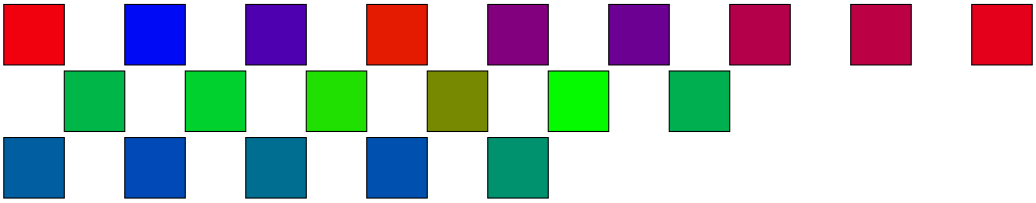


FIGURE 6.10 – Exemple : Critère du saut maximal .

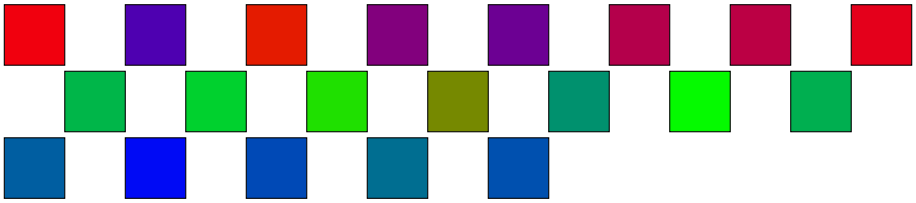


FIGURE 6.11 – Exemple : Lien moyen .

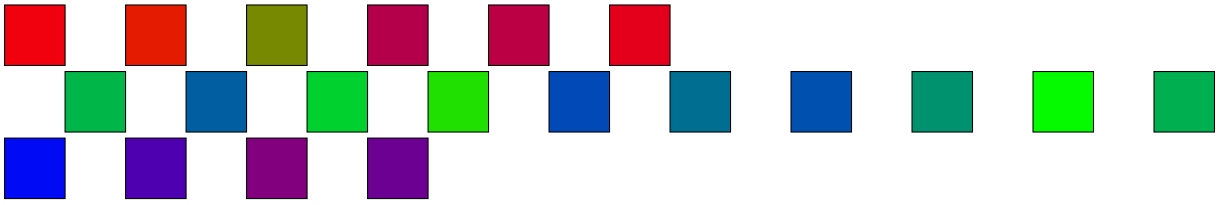


FIGURE 6.12 – Exemple : Lien moyen équilibré.

Pour l'exemple ici illustré, les parcellisations en trois groupes obtenues avec chaque critère d'agglomération sont différentes les unes des autres. Le résultat n'est pas étonnant puisque l'exemple a été construit pour obtenir des résultats différents, illustrant ainsi l'influence du critère d'agglomération sur la parcellisation. Il est cependant possible d'obtenir d'obtenir le même résultat pour différents critères d'agglomération (quitte à permuter le numéro des parcelles).

6.1.2.2 Avantages et inconvénients de certains critères

Comme le montre l'exemple des carrés de couleur, les résultats de la parcellisation varient selon le critère d'agglomération utilisé. De plus, l'exemple illustre l'influence du critère d'agglomération mais ces critères d'agglomération dépendent également de la métrique utilisée. On peut donc se demander si certains critères sont mieux que d'autres.

Tout d'abord, le critère du saut minimal est un critère local puisque la distance entre deux parcelles est donnée par la proximité entre les plus proches éléments. Par conséquent, ne sont pas prises en compte les parcelles plus distantes et la structure globale des parcelles. Le critère du saut minimal vise la séparation mais ne tient pas compte de la compacité des parcelles ni

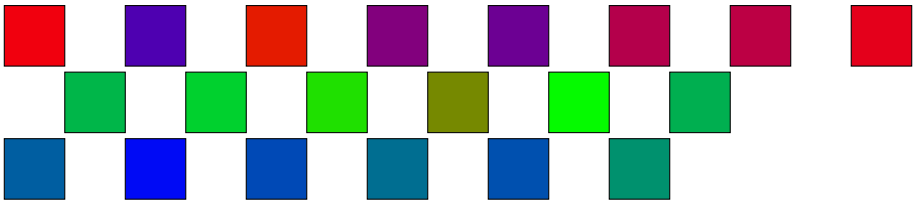


FIGURE 6.13 – Exemple : Centroïde .

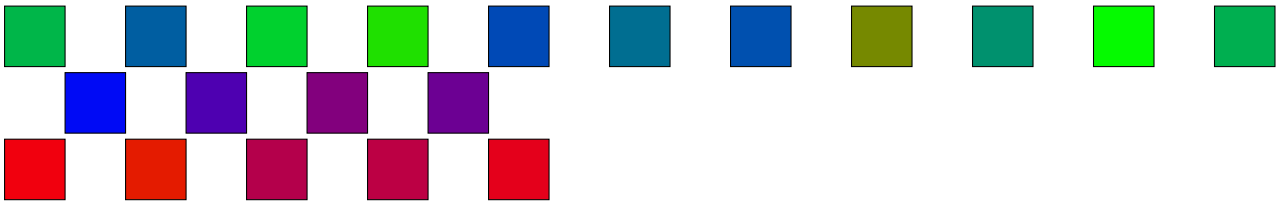


FIGURE 6.14 – Exemple : Médiane .

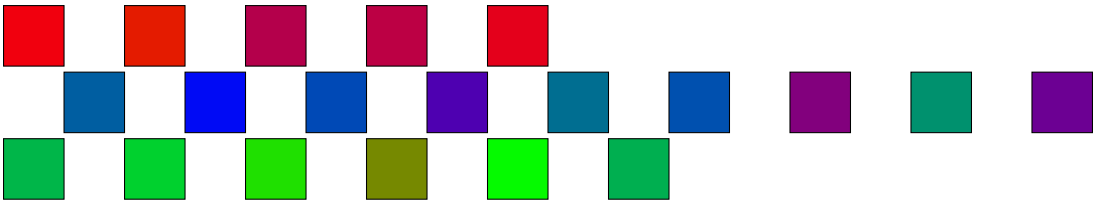


FIGURE 6.15 – Exemple : Critère de Ward .

de l'équilibre entre les parcelles. Comme illustré dans l'exemple, ce critère peut aboutir à la formation de longues chaînes de parcelles [139]. En d'autres termes, le critère du saut minimal tend à ajouter les éléments un par un aux parcelles. En n'imposant aucune contrainte sur la forme des parcelles, ce critère ne permet pas de trouver des parcelles compactes mais autorise la détection de parcelles irrégulières et allongés [181].

À l'inverse du critère du saut minimal, le critère du saut maximal n'est pas un critère local. Il tient compte de l'ensemble de la structure de la parcellisation. Les parcelles ainsi obtenues sont compactes [139] et tendent à avoir le même diamètre (c'est-à-dire la même distance maximale entre deux éléments de la parcelle). Il est en revanche sensible aux éléments aberrants [181].

Le critère du lien moyen évite les inconvénients des critères de saut minimal et maximal en prenant en compte toutes les similarités entre les éléments [139] et apparaît donc comme un compromis entre le critère du saut minimal et le critère du saut maximal. Il tend à joindre des parcelles avec des petites variances et à produire des parcelles avec la même variance [181]. Le lien moyen et la méthode du centroïde peuvent parfois donner des renversements du dendrogramme c'est-à-dire que deux parcelles fusionnées à une itération k sont plus proches que les paires de parcelles fusionnées aux itérations précédentes. Un tel renversement se produit pour notre exemple pour la méthode de la médiane : sur la figure 6.21 on observe en effet qu'à la dernière itération les deux parcelles sont séparées par une distance de 1,02 alors que la parcelle contenant les carrés rouges et la parcelle contenant les carrés violets sont distantes de 1,03.

Le critère de Ward tend à joindre les parcelles avec un petit nombre d'observations et à produire des parcelles avec à peu près le même nombre d'observations. Il est de plus sensible aux observations aberrantes [181]. Si on considère deux paires de parcelles dont les centroïdes sont à la même distance, la méthode de Ward tendra à fusionner les parcelles les plus petites [185]. Le critère de Ward est le critère le plus proche de la méthode des K-moyennes en terme de propriétés et d'efficacité [194].

6.1.3 Formule de mise à jour des distances

Après chaque fusion de deux parcelles, il faut calculer la distance entre la nouvelle parcelle et les autres parcelles. Le calcul de cette distance peut être coûteux puisque pour certains critères, cette distance nécessite la distance entre toutes les paires d'éléments. Cependant, Lance et Williams ont établi une formule pour mettre à jour les dissimilarités après la fusion de deux parcelles [119].

Soient C_i^k et C_j^k (ne contenant éventuellement qu'un seul élément) les deux parcelles à être fusionnées à l'itération k pour former la parcelle $C_{i\cup j}^{k+1}$. La dissimilarité entre la parcelle $C_{i\cup j}^{k+1}$ et une autre parcelle C_l^k (contenant éventuellement un seul élément) qui devient la parcelle C_l^{k+1} (puisqu'elle n'est pas fusionnée) est donnée par

$$d\left(C_{i\cup j}^{k+1}, C_l^{k+1}\right) = \alpha(i)d\left(C_i^k, C_l^k\right)+\alpha(j)d\left(C_j^k, C_l^k\right)+\beta d\left(C_i^k, C_j^k\right)+\gamma\left|d\left((C_i^k, C_l^k\right) - d\left(C_j^k, C_l^k\right)\right| \tag{6.1}$$

Cette formule de mise à jour a l'avantage de n'impliquer que trois distances : la distance entre les deux parcelles fusionnées $d\left(C_i^k, C_j^k\right)$, la distance entre la parcelle C_i^k et la parcelle C_l^k $d\left(C_i^k, C_l^k\right)$ et la distance entre la parcelle C_j^k et la parcelle C_l^k $d\left(C_j^k, C_l^k\right)$. Or chacune de ces distances a déjà été calculée. L'utilisation de la formule de mise à jour permet de ne pas recalculer la distance entre deux parcelles.

Les paramètres $\alpha(i)$, β et γ de la formule dépendent du critère d'agglomération choisi. Initialement, Lance et Williams n'ont pas généralisé la formule de mise à jour au critère de Ward. La formule de mise à jour des distances a été généralisée au critère de Ward par Wishart [225]. La table 6.1 donne la valeur des paramètres $\alpha(i)$, β et γ pour les différents critères d'agglomération.

TABLE 6.1 – Paramètres de la formule (6.1).

Critère d'agglomération	$\alpha(i)$	β	γ
Critère du saut minimal	0,5	0	-0,5
Critère du saut maximal	0,5	0	0,5
Lien moyen	$\frac{\lvert C_i^k\rvert}{\lvert C_i^k\rvert+\lvert C_j^k\rvert}$	0	0
Méthode du centroïde	$\frac{\lvert C_i^k\rvert}{\lvert C_i^k\rvert+\lvert C_j^k\rvert}$	$-\frac{\lvert C_i^k\rvert\lvert C_j^k\rvert}{(\lvert C_i^k\rvert+\lvert C_j^k\rvert)^2}$	0
Méthode de Ward	$\frac{\lvert C_i^k\rvert+\lvert C_l^k\rvert}{\lvert C_i^k\rvert+\lvert C_j^k\rvert+\lvert C_l^k\rvert}$	$-\frac{\lvert C_l^k\rvert}{\lvert C_i^k\rvert+\lvert C_j^k\rvert+\lvert C_l^k\rvert}$	0
Méthode de la médiane	0.5	-0.25	0

6.1.4 Représentation de la parcellisation

Le résultat d'un clustering hiérarchique agglomératif se représente sous forme d'un arbre, appelé dendrogramme. Le dendrogramme se construit en traçant un trait entre les deux parcelles fusionnées à chaque itération. Certaines représentations font apparaître comme ordonnée la distance à laquelle les parcelles ont été fusionnées.

Les figures 6.16 à 6.22 montrent le dendrogramme obtenu pour chacun des critères d'agglomération appliqué à l'exemple 6.1.2.1. Sur les figures, l'échelle verticale correspond à la distance à laquelle ont été fusionnées les parcelles. Sur la figure 6.21 est observable le renversement des distances qui peut survenir pour la méthode de la médiane et du centroïde. En effet, à la dernière itération les deux parcelles fusionnées (cluster à droite avec les carrés verts et bleus et cluster à gauche avec les carrés rouges et violets) sont plus proches que les parcelles fusionnées à l'itération précédente (parcelle contenant les carrés rouges et parcelle contenant les carrés violets). La figure 6.16 illustre l'effet de chaînage du lien par saut minimal notamment pour les carrés verts.

Le dendrogramme est un outil pratique visuel pour étudier la parcellisation puisqu'on y voit la structure. Cependant à partir d'un certain nombre de vecteurs à regrouper, le dendrogramme ne tient plus sur une feuille A4 et devient donc difficile à dessiner.

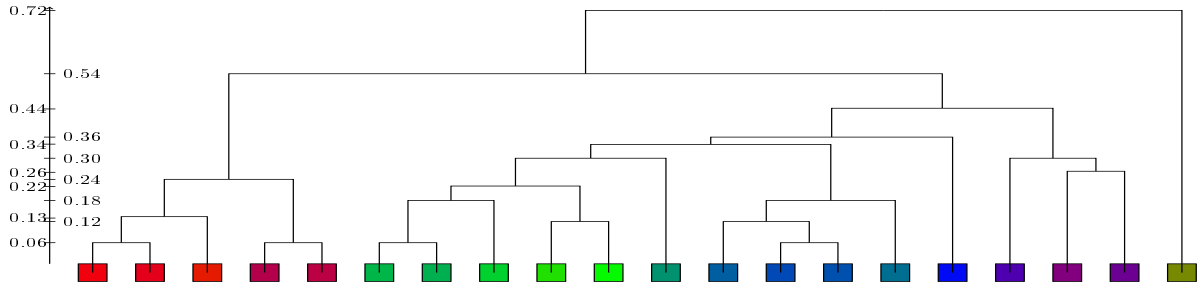


FIGURE 6.16 – Exemple : Dendrogramme avec le critère du saut minimal.

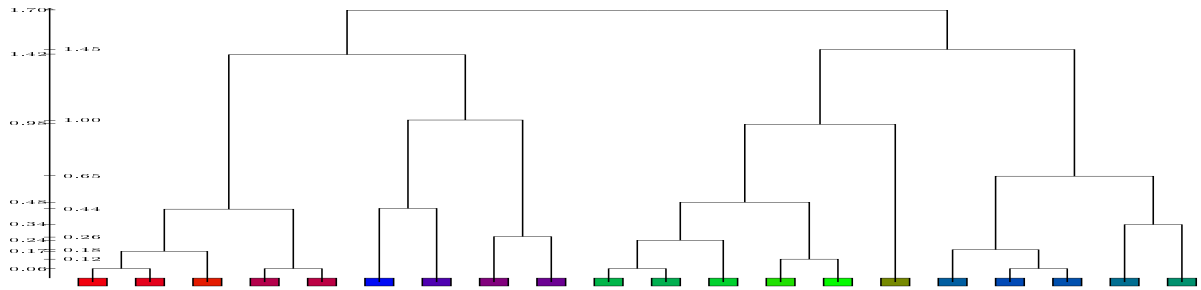


FIGURE 6.17 – Exemple : Dendrogramme avec le critère du saut maximal.

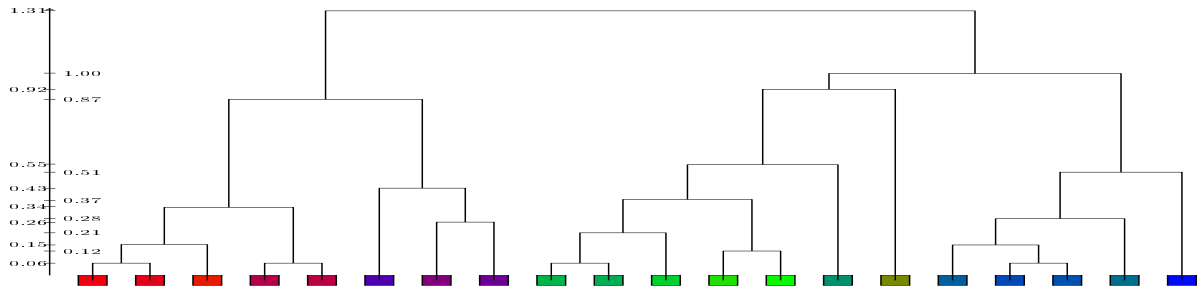


FIGURE 6.18 – Exemple : Dendrogramme avec le lien moyen.

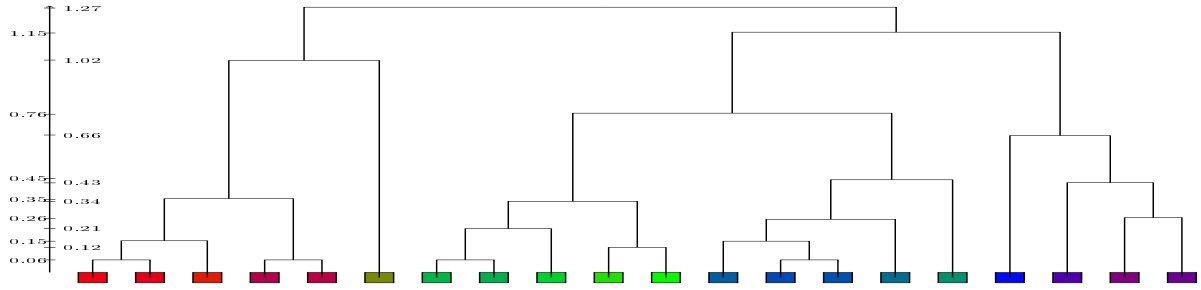


FIGURE 6.19 – Exemple : Dendrogramme avec le lien moyen équilibré.

6.1.5 Comment déterminer le nombre de parcelles ?

Lors du découpage de l'exemple en trois parcelles, les résultats montrés en 6.1.2.1 suggèrent qu'un découpage en trois parcelles pour séparer le vert, le rouge et le bleu n'est peut-être pas le

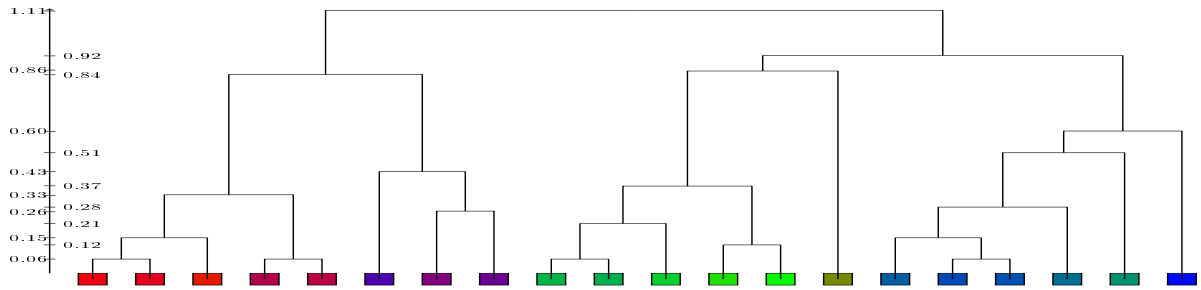


FIGURE 6.20 – Exemple : Dendrogramme avec la méthode du centroïde.

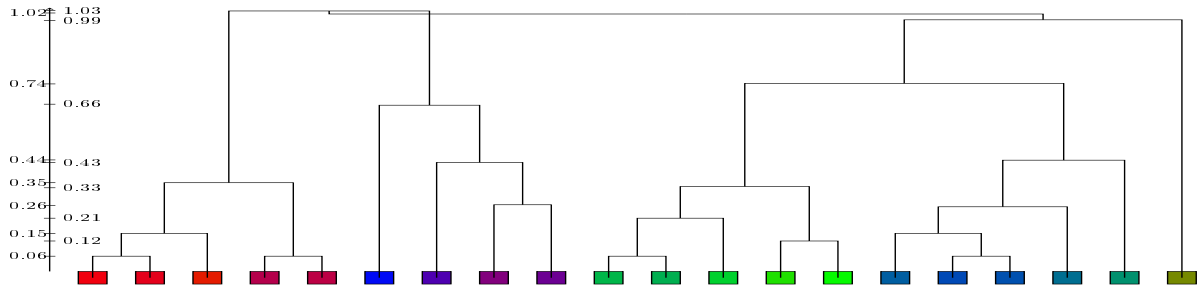


FIGURE 6.21 – Exemple : Dendrogramme avec la méthode de la médiane.

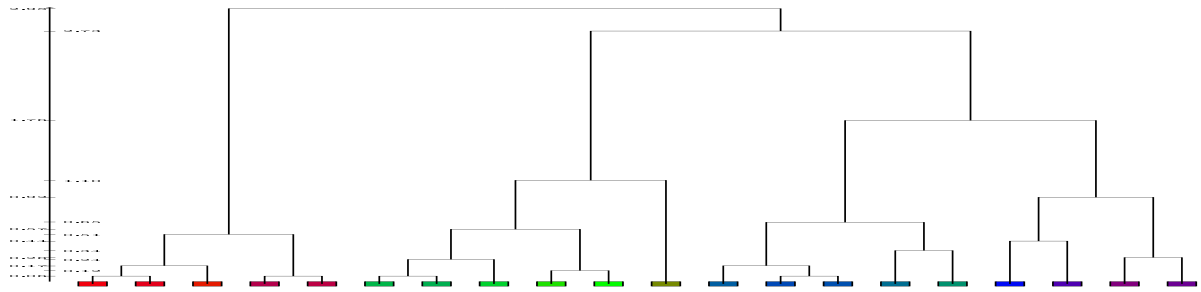


FIGURE 6.22 – Exemple : Dendrogramme avec le critère de Ward.

plus approprié. Pour cet exemple, on pourrait imaginer un découpage en quatre parcelles : une pour les carrés bleus, une pour les carrés rouges, une pour les carrés verts et une pour les carrés violets.

Bien qu'il existe plusieurs critères, déterminer le nombre optimal de parcelles pour représenter l'ensemble $\boldsymbol{X}$ reste un problème difficile et non résolu car le nombre optimal de parcelles dépend à la fois de la méthode de parcellisation utilisée et des caractéristiques de la distributions des vecteurs [72]. Sont présentés ici différents critères utilisés pour déterminer le nombre optimal de parcelles.

6.1.5.1 Indice de silhouette

L'indice de silhouette [176] mesure la qualité de la parcellisation en mesurant pour chaque vecteur à quel point il correspond à la parcelle à laquelle il a été assigné. Pour chaque vecteur, il compare la dissimilarité intra-parcelle à la dissimilarité inter-parcelles. L'indice de silhouette évalue la compacité des parcelles et leur degré de séparation.

Soit $\boldsymbol{x}_j$ un vecteur et soit C_i^k la parcelle à laquelle il a été assigné, on définit la dissimilarité intra-parcelle par

$$a_j^k = \frac{1}{|C_i^k|} \sum_{\boldsymbol{x}_l \in C_i^k} \delta(\boldsymbol{x}_j, \boldsymbol{x}_l) \ .$$

La dissimilarité inter-parcelles est définie par

$$b_j^k = \min_{m \neq i} \left(\frac{1}{|C_m^k|} \sum_{\boldsymbol{x}_l \in C_m^k} \delta(\boldsymbol{x}_j, \boldsymbol{x}_l) \right) \ .$$

L'indice de silhouette pour le vecteur $\boldsymbol{x}_j$ est alors donné par

$$s_j^k = \frac{b_j^k - a_j^k}{\max(a_j^k, b_j^k)} \ .$$

Pour un niveau de parcellisation k, l'indice de silhouette est défini par

$$s(k) = \frac{1}{n_{\mathrm{V}}} \sum_{j=1}^{n_{\mathrm{V}}} s_j^k \ .$$

L'indice de silhouette s_j^k est compris entre -1 et 1. Lorsqu'il vaut 1, cela signifie que la dissimilarité intra-parcelle est beaucoup plus petite que la plus petite des dissimilarités inter-parcelles donc le vecteur $\boldsymbol{x}_j$ a été assigné dans la parcelle appropriée. Si s_j^k est égale à -1, la dissimilarité intra-parcelle est plus grande que la plus petite dissimilarité inter-parcelles donc $\boldsymbol{x}_j$ est plus près de la parcelle pour lequel le minimum de dissimilarité inter-parcelles a été atteint que de la parcelle à laquelle il a été assigné. Si $s_i^k = 0$, le vecteur $\boldsymbol{x}_j$ est à la même distance de la parcelle à laquelle il a été assignée que de la parcelle pour lequel le minimum de dissimilarité inter-parcelles a été atteint.

Le nombre optimal de parcelles n_{C}^* est alors donné par

$$n_{\mathrm{C}}^* = \underset{k \in \{2,\ldots,n_{\mathrm{V}}\}}{\arg\max} \ s(k) \ . \tag{6.2}$$

Fujita *et al.* ont montré que cette procédure fonctionne lorsque les parcelles sont homogènes c'est-à-dire qu'elles ont la même variabilité interne ; mais que la procédure échoue lorsqu'il y a une parcelle avec une variabilité interne dominante.

En général, le calcul du coefficient de silhouette est simplifié car il nécessite le calcul de la matrice de distances qui est coûteux lorsque le nombre n_{V} de vecteurs à parcelliser augmente. Par exemple, Arslan *et al.* proposent de calculer le coefficient b_j^k en considérant uniquement les parcelles voisines de la parcelle considérée [12]. Une autre manière de simplifier le calcul est de calculer a_j^k et b_j^k non pas en fonction des distances entre les vecteurs mais en fonction de la distance par rapport au centroïde des parcelles [219]. L'indice de silhouette simplifié ssi_j devient alors :

$$ssi_j^k = \frac{b_j'^k - a_j'^k}{\max(a_{'j}^k, b_{'j}^k)} \ ,$$

avec

$$a_j'^k = \delta(\boldsymbol{x}_j, \overline{C_i^k}) \quad \text{et} \quad b_j'^k = \min_{m \neq i} \left(\delta(\boldsymbol{x}_j, \overline{C_m^k}) \right) \ .$$

6.1.5.2 Méthode du coude

On définit la somme des carrés intra-parcelle par

$$W_k = \sum_{i=1}^{k} \sum_{\boldsymbol{x}_{j_1} \in C_i^k} \sum_{\substack{\boldsymbol{x}_{j_2} \in C_i^k \\ j_2 \neq j_1}} (\boldsymbol{x}_{j_1} - \boldsymbol{x}_{j_2})(\boldsymbol{x}_{j_1} - \boldsymbol{x}_{j_2})^\top \ . \tag{6.3}$$

La méthode du coude consiste à déterminer le nombre optimal de parcelles n_{C}^* graphiquement en traçant W_k en fonction de k. n_{C}^* correspond au nombre de parcelle pour lequel on observe un coude donc un changement de pente comme illustré en figure 6.23.

FIGURE 6.23 – Méthode du coude [64].

6.1.5.3 Indice de Calinski et Harabaz

Notons k le nombre de parcelles choisies pour parcelliser l'ensemble $\boldsymbol{X}$ comportant n_{V} éléments. On définit B_k la somme des carrés inter-parcelles par

$$B_k = \sum_{i=1}^{k} \left|C_i^k\right| (\overline{C_i^k} - \overline{\boldsymbol{X}})(\overline{C_i^k} - \overline{\boldsymbol{X}})^\top \ ,$$

avec $\overline{\boldsymbol{X}}$ la moyenne des éléments de $\boldsymbol{X}$, et la somme des carrés intra-parcelle définie par (6.3).

L'indice de Calinski et Harabaz est défini par

$$CH(k) = \frac{\frac{B_k}{k-1}}{\frac{W_k}{n_{\mathrm{V}}-k}} \ .$$

Le nombre optimal n_{C}^* est donné par

$$n_{\mathrm{C}}^* = \underset{k\in\{2,\dots,n_{\mathrm{V}}\}}{\arg\max}\ CH(k) \ . \tag{6.4}$$

6.1.5.4 Indice de Krzanowski et Lai

Rappelons que notre ensemble $\boldsymbol{X}$ a n_{V} éléments de taille n_{T}.

On définit à partir de la somme des carrés intra-parcelle donnée en (6.3) :

$$DIFF(k) = (k-1)^{\frac{2}{n_{\mathrm{T}}}} W_{k-1} - k^{\frac{2}{n_{\mathrm{T}}}} W_k \ .$$

On définit ensuite l'indice de Krzanowski et Lai par

$$KL(k) = \left|\frac{DIFF(k)}{DIFF(k+1)}\right| \ .$$

Le nombre optimal de parcelles n_{C}^* est donné par

$$n_{\mathrm{C}}^* = \underset{k\in\{2,\dots,n_{\mathrm{V}}\}}{\arg\max}\ KL(k) \ . \tag{6.5}$$

6.1.5.5 Indice du saut (*gap statistic*)

L'idée de l'indice du saut introduit par Tibshirani est de standardiser le graphe de $\log(W_k)$ avec W_k la somme des carrés intra-parcelles définie en (6.3) en le comparant à l'espérance sous une distribution de référence (où les vecteurs seraient issus d'une même distribution). Notons $E^*_{n_\mathrm{V}}$ l'espérance sous la distribution de référence. L'indice du saut est alors défini par

$$Gap_{n_\mathrm{V}}(k) = E *_{n_\mathrm{V}} \log(W_k) - \log(W_k) \ .$$

Le nombre optimal de parcelles n^*_C est donné par

$$n^*_\mathrm{C} = \underset{k\in\{2,\dots,n_\mathrm{V}\}}{\arg\max}\ Gap_{n_\mathrm{V}}(k) \ . \tag{6.6}$$

6.1.5.6 *Slope statictic*

Fujita *et al.* ont défini la *slope statictic* à partir de l'indice de silhouette introduit en section 6.1.5.1. Leur idée est de choisir un nombre de parcelle n^*_C tel que les n^*_C parcelles obtenues ne peuvent pas être scindées en parcelles plus petites sans qu'il y ait une diminution de l'indice de silhouette [72].

Ils déterminent le nombre optimal de parcelles n^*_C par

$$n^*_\mathrm{C} = \underset{k\in\{2,\dots,n_\mathrm{V}-1\}}{\arg\max}\ -(s(k+1)-s(k))s(k)^p \ , \tag{6.7}$$

où p est un paramètre ajustable pour choisir entre un critère où le saut $(s(k+1)-s(k))$ est plus important (pour de petits p) et un critère où la valeur de l'indice de silhouette $s(k)$ a plus de poids (pour de larges p). Ils ont montré que quelle que soit la configuration des données, $p = 1$ donne de bons résultats.

6.1.5.7 Exemple

Appliquons quelques-uns de ces critères à l'exemple 6.1.2.1 en utilisant la distance de Ward. La figure 6.24 montre trois indices : à gauche l'indice de silhouette, au milieu l'indice de Calinski et Harabaz et à droite la slope statistic. Déterminons maintenant le nombre optimal de parcelles grâce à ces différents indices. L'indice de silhouette donne $n^*_\mathrm{C} = 9$. L'indice de Calinski et Harabaz est maximal pour $n^*_\mathrm{C} = 16$. La slope statistic avec p=0 donne $n^*_\mathrm{C} = 16$. Si on prend p=1, on obtient $n^*_\mathrm{C} = 10$. Passons à p=2, on détermine alors $n^*_\mathrm{C} = 9$. Pour p=3, on obtient également $n^*_\mathrm{C} = 9$. Suivant l'indice employé, on obtient donc un nombre optimal de parcelles différent. L'exemple illustre bien la difficulté à déterminer le nombre optimal de parcelles alors qu'on ne considère que vingts objets avec trois caractéristiques chacun. On note toutefois que plusieurs indices donnent $n^*_\mathrm{C} = 9$ ou $n^*_\mathrm{C} = 16$ qui pourraient donc faire consensus. Il faut cependant remarquer qu'on a utilisé la distance de Ward qui ne donne pas le meilleur dendrogramme, ce qui pourrait expliquer l'absence de consensus pour le nombre optimal de parcelles.

Néanmoins, pour trouver le nombre de parcelles optimal, tous ces indices requièrent de tester tous les niveaux de parcellisation possibles. Or, si on regroupe n_V vecteurs, il existe n_V niveaux de parcellisation possibles : du niveau le plus fin où chaque vecteur est dans sa propre parcelle au niveau le plus grossier où les vecteurs appartiennent à une seule et même parcelle. Le coût pour trouver le nombre optimal de parcelles grandit donc avec le nombre n_V d'éléments à parcelliser et peut vite devenir prohibitif.

6.2 Algorithme de Ward

Suivant l'étude bibliographique et en particulier l'article de Thirion *et al.* [202], nous avons choisi de développer l'algorithme de Ward puisqu'il donne de meilleurs résultats et a une meilleure

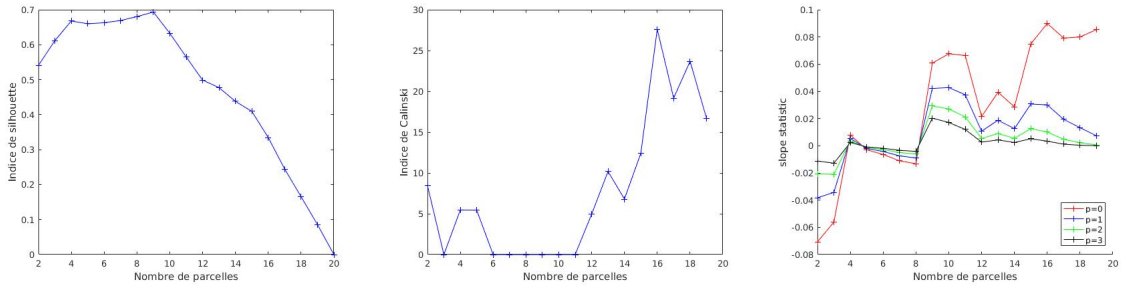


FIGURE 6.24 – Détermination du nombre optimal de parcelles pour l'exemple des carrés de couleur.

reproductibilité sur notre cas d'application à savoir les données d'imagerie fonctionnelle. Thirion *et al.* ont implémenté l'algorithme de Ward en Python dans la librairie Nilearn [164, 3]. Afin de réduire le coût du calcul de la distance pour de très grandes données (comme les données d'IRMf), ils proposent une version avec contraintes spatiales n'autorisant que le regroupement de voxels voisins spatialement. Un tel regroupement est illustré en figure 6.25.

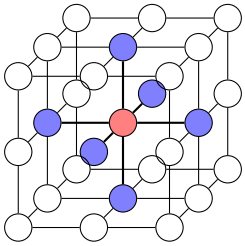


FIGURE 6.25 – Voisinage d'un point dans $\mathbb{R}^3$.

À la différence de l'algorithme présenté par Thirion *et al.*, nous avons développé un algorithme de Ward sans contrainte spatiale qui puisse traiter un grand nombre de données. En effet, sur les données d'un sujet, l'algorithme de Nilearn ne peut pas traiter sans contraintes spatiales plus de 40 000 voxels sur COMP1 (**cf. section 8.1**).

Ci-dessous est rappelé l'algorithme de Ward. La deuxième sous-partie est consacrée au calcul de la distance. Suit l'implémentation de l'algorithme et l'évaluation des complexités en mémoire et en nombre d'opération flottante.

6.2.1 Principe

L'algorithme de Ward est un algorithme de clustering hiérarchique agglomératif, qui utilise le critère d'agglomération de Ward.

6.2.1.1 Métrique et critère d'agglomération pour l'algorithme de Ward

Nous souhaitons regrouper les n_{V} éléments $(\boldsymbol{x}_j, j \in \{1, \ldots, n_{\mathrm{V}}\})$ ayant n_{T} caractéristiques. Pour l'algorithme de Ward, la métrique choisie pour calculer la distance entre deux éléments $\boldsymbol{x}_j$ et $\boldsymbol{x}_i$ est la distance euclidienne au carré :

$$\delta(\boldsymbol{x}_i, \boldsymbol{x}_j) = \|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2^2 = \sum_{t=1}^{n_{\mathrm{T}}} (x_{ti} - x_{tj})^2 \ . \tag{6.8}$$

Le critère d'agglomération choisi est le critère de Ward qui minimise l'augmentation de la somme des carrés. Soient C_i^k et C_i^k deux parcelles, la distance entre C_i^k et C_j^k est donnée par

$$d(C_i^k, C_j^k) = \frac{\left|C_i^k\right| \left|C_j^k\right|}{\left|C_i^k\right| + \left|C_j^k\right|} \delta\left(\overline{C_i^k}, \overline{C_j^k}\right) \tag{6.9}$$

6.2.1.2 Algorithme

La figure 6.2 montre le déroulement de l'algorithme. Décrivons cet algorithme en détail.

1. Pour chaque paire d'éléments, calculer la distance entre les vecteurs
2. Tant que toutes les parcelles ne sont pas regroupées en une seule parcelle
   a) Trouver les deux parcelles séparées par la distance minimale
   b) Fusionner les deux parcelles
   c) Mettre à jour la matrice de distances

En se référant à l'équation 6.9, on constate qu'à chaque étape il faut donc calculer le centroïde de la nouvelle parcelle issue de la fusion des deux parcelles pour mettre à jour la distance entre la nouvelle parcelle et les autres parcelles. Cependant, la formule de mise à jour de Lance-Williams exprime la distance entre la nouvelle parcelle $C_{i\cup j}^{k+1}$ et une autre parcelle C_l^{k+1} à l'aide de la distance entre la parcelle C_i^k et la parcelle C_l^k et de la distance entre la parcelle C_j^k et la parcelle C_l^k. Comme la parcelle C_l^k n'a pas été fusionnée à l'itération $k+1$, elle est égale à elle-même à l'itération $k+1$ ($C_l^k = C_l^{k+1}$). La formule de mise à jour de lance-Williams pour le critère de Ward (démontrée en Annexe B) est donnée par

$$\begin{aligned} d\left(C_{i\cup j}^{k+1}, C_l^{k+1}\right) = & \frac{\left|C_i^k\right| + \left|C_l^k\right|}{\left|C_i^k\right| + \left|C_j^k\right| + \left|C_l^k\right|} d\left(C_i^k, C_l^k\right) + \frac{\left|C_j^k\right| + \left|C_l^k\right|}{\left|C_i^k\right| + \left|C_j^k\right| + \left|C_l^k\right|} d\left(C_j^k, C_l^k\right) \\ & - \frac{\left|C_l^k\right|}{\left|C_i^k\right| + \left|C_j^k\right| + \left|C_l^k\right|} d\left(C_i^k, C_j^k\right) \end{aligned} \tag{6.10}$$

À partir de l'équation 6.10, on constate qu'une fois la matrice de distances entre les éléments calculée ($\boldsymbol{x}_j, j \in \{1, \ldots, n_{\mathrm{V}}\}$) à partir de la métrique δ, tous les calculs peuvent se faire à partir de la matrice de distances $\boldsymbol{D}$, sans avoir à utiliser la matrice de données $\boldsymbol{X}$. L'algorithme de Ward pourrait donc ne prendre en entrée que la matrice de distances $\boldsymbol{D}$ sans connaître les données $\boldsymbol{X}$.

Toutefois, pour notre cas d'application, nous aurons besoin de calculer la matrice de distances entre les éléments. C'est pourquoi le calcul de la distance est inclus dans l'algorithme de Ward. L'algorithme 5 reprend les grandes étapes de l'algorithme.

Algorithme 5 : Algorithme de Ward simplifié

```
pour chaque élément v∈ {1,...,n_V} faire
    pour Pour chaque élément v'≠v faire
        Calculer la distance d(X_v, X_v') = (1/2) ||X_v − X_v'||_2^2
pour iter de 1 à n_V-1 faire
    (i*,j*)=arg min_{i,j≠i} d(i,j)
    Fusionner les clusters (i*,j*)
    Mettre à jour la matrice des distances pour le nouveau cluster
```

n_T	n_V	Distance	Part	Itérations	Part	Total
1 000	10 000	42,27	83,97%	8,07	16,03%	50,34
	30 000	378,19	81,87%	83,75	18,13%	461,95
	50 000	1 049,69	81,19%	243,23	18,81%	1 292,93
	70 000	2 057,43	80,83%	487,82	19,17%	2 545,28
	100 000	4 199,21	80,41%	1 023,14	19,59%	5 222,43
	120 000	6 049,13	79,69%	1 541,89	20,31%	7 591,15
10 000	10 000	448,96	98,13%	8,56	1,87%	457,52
	30 000	4 038,77	97,83%	89,57	2,17%	4 128,35
	50 000	11 218,68	97,72%	262,13	2,28%	11 480,84
	70 000	22 376,47	97,71%	523,28	2,29%	22 899,79
	10 0000	44 918,72	97,44%	1 180,98	2,56%	46 099,84
20 000	10 000	919,79	99,07%	8,64	0,93%	928,43
	30 000	8 285,79	98,92%	90,28	1,08%	8 376,08
	50 000	23 006,90	98,87%	262,47	1,13%	23 269,39
	70 000	45 110,97	98,84%	528,64	1,16%	45 639,64
	100 000	91 480,48	98,80%	1 113,73	1,20%	92 594,34

TABLE 6.2 – Répartition du temps de calcul entre le calcul de la distance et les itérations .

6.2.2 Calcul de la matrice de distances

L'une des étapes coûteuses de l'algorithme de Ward est le calcul de la matrice de distances $\boldsymbol{D}$. En effet, nous devons calculer la distance entre chaque paire de vecteurs $(\boldsymbol{x}_i, \boldsymbol{x}_j), i \in \{1, \ldots, n_\mathrm{V}\}, j \in \{1, \ldots, n_\mathrm{V}\}$. Pour les paires $(\boldsymbol{x}_j, \boldsymbol{x}_j), j \in \{1, \ldots, n_\mathrm{V}\}$ la distance vaut 0 et n'a donc pas besoin d'être calculée. De plus, il ne faudra pas tenir compte de cette distance lorsqu'on calculera le minimum des distances sur toutes les paires. Pour chaque paire $(\boldsymbol{x}_i, \boldsymbol{x}_j), i \in \{1, \ldots, n_\mathrm{V}\}, j \in \{1, \ldots, n_\mathrm{V}\}$, par symétrie de la distance une seule distance doit être calculée $\delta(\boldsymbol{x}_i, \boldsymbol{x}_j) = \delta(\boldsymbol{x}_j, \boldsymbol{x}_i)$. Nous devons donc calculer $n_\mathrm{V}(n_\mathrm{V} - 1)/2$distances. L'expression de la distance euclidienne est donnée par l'équation 6.8 et nécessite donc $3n_\mathrm{T}$ opérations (une soustraction et une multiplication pour chaque caractéristique puis la somme de toutes les caractéristiques). La complexité temporelle du calcul de la distance est donc en $O(n_\mathrm{T} n_\mathrm{V}^2/2)$.

La table 6.2 montre les temps de calcul en secondes de la distance et des itérations, c'est-à-dire la fusion des parcelles deux à deux jusqu'à ce qu'il ne reste qu'une parcelle ; ainsi que la proportion de chacune des parties sur le temps complet de l'algorithme. On constate que pour $n_\mathrm{T} = 1\,000$ qui correspond à la parcellisation des données pour un sujet, le calcul de la distance prend 80% du temps total. Mais lorsqu'on passe à $n_\mathrm{T} \geq 10\,000$, qui correspond à la parcellisation des données concaténées de plusieurs sujets, le calcul de la distance prend environ 98% du temps de calcul total. Ce constat explique que nous ayons concentré notre travail sur l'amélioration du calcul de la distance. Cet effort est détaillé dans le chapitre 7.

6.2.3 Implémentation de l'algorithme

L'algorithme complet est donné en annexe D. La section 6.2.3.1 explique ce que doit renvoyer l'algorithme de Ward afin d'obtenir la parcellisation pour n'importe quelle échelle. La section 6.2.3.2 montre comment mettre à jour la distance et comment réduire le temps de calcul pour trouver la distance minimale à chaque itération.

6.2.3.1 Comment obtenir la répartition des vecteurs dans les parcelles ?

L'algorithme prend en entrée la matrice de données $\boldsymbol{X}$ de taille $n_\mathrm{T} \times n_\mathrm{V}$ contenant les n_V vecteurs de n_T caractéristiques à regrouper.

Intéressons-nous maintenant à ce que doit retourner l'algorithme pour qu'on puisse obtenir la parcellisation à n'importe quelle échelle. On pourrait garder à chaque itération à quelle parcelle appartient l'ensemble des vecteurs. Cependant, cela requerrait une matrice de taille $n_{\mathrm{V}}-1\times n_{\mathrm{V}}$, ce qui serait aussi coûteux que la matrice de distances. Il faut donc trouver une autre stratégie moins coûteuse en espace mémoire. De plus à chaque itération, seules deux parcelles sont fusionnées et donc changent d'indice de parcelle. On peut donc conserver à chaque itération uniquement le couple d'indices des deux parcelles qui ont été fusionnées ensemble.

L'algorithme renverra donc la matrice $\boldsymbol{mergedClusters}$ de taille $2 \times (n_{\mathrm{V}} - 1)$ telle que $\boldsymbol{mergedClusters}_{iter}$ contient l'indice des parcelles fusionnées à l'étape $iter$. Reste à donner un indice aux parcelles fusionnées. Les n_{V} colonnes de la matrice $\boldsymbol{X}$ forment les n_{V} parcelles initiales. La parcelle nouvellement formée à l'itération k a pour numéro $n_{\mathrm{V}} + k$.

La matrice $\boldsymbol{mergedClusters}$ suffit-elle à obtenir toutes les parcellisations ? Prenons un exemple avec $n_{\mathrm{V}} = 10$ éléments et considérons le dendrogramme présenté en figure 6.26.

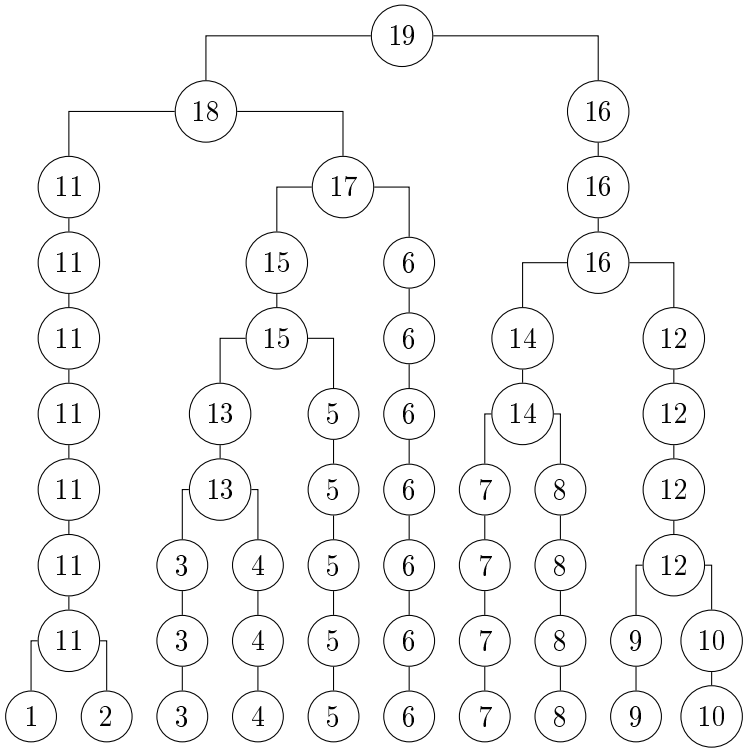


FIGURE 6.26 – Exemple de dendrogramme pour 10 éléments à parcelliser.

La matrice $\boldsymbol{mergedClusters}$ correspondant à la figure 6.26 s'écrit :

$$\boldsymbol{mergedClusters} = \begin{pmatrix} 1 & 9 & 3 & 7 & 13 & 14 & 6 & 11 & 16 \\ 2 & 10 & 4 & 8 & 6 & 12 & 15 & 17 & 18 \end{pmatrix} .$$

Comment à partir de cette matrice trouver la décomposition en n_{C} parcelles ? Supposons qu'on veuille former n_{V}-1 parcelles, cela veut dire que seulement deux parcelles initiales ont été fusionnées, autrement dit une seule itération de l'algorithme a été réalisé. Or $\boldsymbol{mergedClusters}_1$ donne les deux parcelles fusionnées à la première étape, c'est-à-dire pour obtenir les $n_{\mathrm{V}} - 1$ parcelles. Supposons maintenant qu'on veuille $n_{\mathrm{V}}-2$ parcelles c'est-à-dire qu'on a effectué deux itérations de l'algorithme, il nous faut alors regarder $\boldsymbol{mergedClusters}_1$ et $\boldsymbol{mergedClusters}_2$ qui identifient quelles ont été les parcelles fusionnées aux étapes 1 et 2. Par récurrence, on peut montrer que pour obtenir $n_{\mathrm{V}} - k$ parcelles, il faut regarder dans $\boldsymbol{mergedClusters}_j, j \in \{1, \ldots, k\}$ pour savoir quelles parcelles ont été fusionnées aux étapes $j \in \{1, \ldots, k\}$ pour former $n_{\mathrm{V}} - k$ parcelles. Nous souhaitons obtenir un nombre arbitraire n_{C} de parcelles, en suivant le raisonnement précédent, on sait que les parcelles fusionnées pour obtenir n_{C} parcelles sont données par

$\boldsymbol{mergedClusters}_j$ pour $j \in \{1, \ldots, n_\mathrm{V} - n_\mathrm{C}\}$. Mais, cela n'indique pas quels sont les indices des parcelles présentes c'est-à-dire quelles sont les parcelles formant les n_C parcelles recherchées. Or ces parcelles présentent à l'étape n_V-n_C seront fusionnées aux itérations suivantes. Les indices de ces parcelles sont donc présents dans les colonnes n_V-n_C+1 à n_V-1 de $\boldsymbol{mergedClusters}$ puisqu'elles sont fusionnées à partir de l'itération n_V-n_C+1 jusqu'à la dernière itération. Le problème est que dans les colonnes d'indices n_V-n_C+1 à n_V-1 de $\boldsymbol{mergedClusters}$, on dispose de $2(n_\mathrm{C} - 1)$ indices de parcelles (puisqu'il y a $n_\mathrm{C} - 1$ colonnes et chaque colonne contient deux indices de parcelles). Cela vient du fait que les parcelles présentes à l'étape n_V-n_C seront combinées aux étapes ultérieures pour former de nouvelles parcelles. Or, on a adopté la convention qu'à l'étape k, la nouvelle parcelle formée a pour indice $n_\mathrm{V} + k$. De fait, à l'étape n_V-n_C, les parcelles présentent peuvent avoir pour indice maximal n_V+n_V-n_C. Pour obtenir n_C parcelles, il faut donc regarder les colonnes d'indices n_V-n_C+1 à n_V-1 de $\boldsymbol{mergedClusters}$ et ne garder que les parcelles dont les indices sont inférieurs à n_V+n_V-n_C.

Reprenons notre exemple et supposons qu'on veuille avoir n_C=4 parcelles. La technique décrite précédemment donne qu'il faut retenir les parcelles n^{os} 6, 11, 15 et 16. Reste maintenant à déterminer dans laquelle de ces parcelles chaque vecteur initial se range. Le dendrogramme présenté en figure 6.26 donne visuellement la répartition des vecteurs initiaux. Le dendrogramme n'étant représentable que pour un petit nombre de vecteurs initiaux, notre objectif est de trouver la parcellisation à partir de $\boldsymbol{mergedClusters}$. Pour cela pour chaque vecteur, il faut trouver où il apparaît dans $\boldsymbol{mergedClusters}$. Supposons que le vecteur $\boldsymbol{x}_j$ qui porte donc le numéro j apparaisse dans la $l^{ème}$ colonne de $\boldsymbol{mergedClusters}$. Cela veut dire que le vecteur $\boldsymbol{x}_j$ appartient maintenant à la parcelle $n_\mathrm{V} + l$. Si cette parcelle n'est pas une des parcelles sélectionnées, il faut de nouveau chercher où elle apparaît dans $\boldsymbol{mergedClusters}$ et ce jusqu'à trouver une parcelle sélectionnée. Sur notre exemple, le vecteur 1 appartient ainsi à la parcelle 11, qui a été sélectionnée, de même pour le vecteur 2. Prenons maintenant le vecteur 3, il appartient à la parcelle 13, qui elle n'a pas été sélectionnée. Cherchons donc dans quelle parcelle a été fusionnée la parcelle 13. La parcelle 13 a été fusionnée dans la parcelle 15, qui est une des parcelles choisies.

La matrice $\boldsymbol{mergedClusters}$ est donc suffisante pour trouver la parcellisation. En revanche, l'obtention de la parcellisation à échelle donnée est coûteuse car pour chaque vecteur initial, il faut parcourir plusieurs fois la matrice $\boldsymbol{mergedClusters}$. En effet, pour un vecteur initial $\boldsymbol{x}_j$ on parcourt une première fois $\boldsymbol{mergedClusters}$ jusqu'à trouver la colonne l_1 où j apparaît. Deux cas sont alors possibles : soit la parcelle $l_1 + n_\mathrm{V}$ est un des indices des parcelles choisies et on finit, soit la parcelle $l_1 + n_\mathrm{V}$ n'a pas été sélectionnée. Dans le deuxième cas, il faut de nouveau chercher dans $\boldsymbol{mergedClusters}$ où l'indice l_1 apparaît en commençant à la colonne l_1+1 puisque la parcelle $n_\mathrm{V}+l_1$ ne peut pas être fusionnée avant d'avoir été créée à l'itération l_1. La figure 6.27 illustre le processus itératif expliqué ci-dessus. Dans le pire cas, pour un vecteur, la complexité temporelle est en $O(n_\mathrm{V}^2)$. Supposons qu'on a dix vecteurs numérotés de un à dix dont la parcellisation forme une chaîne c'est-à-dire que les vecteurs 1 et 2 sont dans la parcelle 11 puis qu'on ajoute les vecteurs 3 à 10 à cette parcelle 11 l'un après l'autre, lorsqu'on cherche à quelle parcelle appartient le vecteur 1 pour un découpage en trois parcelles, on obtient la parcelle 11. Or la parcelle 11 appartient à la parcelle 12 ainsi de suite jusqu'à être dans la parcelle 17. Et à chaque étape i, on parcourt la matrice $\boldsymbol{mergedClusters}$ à partir de la case i. Or, il faut traiter n_V vecteurs donc dans le pire des cas, la complexité est en $O(n_\mathrm{V}^3)$.

Pour y remédier, l'idée est de conserver pour chaque parcelle dans quelle parcelle parente elle a été fusionnée. Pour cela, on utilise un vecteur $\boldsymbol{parent}$ de taille $2n_\mathrm{V} - 1$ tel que $\boldsymbol{parent}_j$ contienne le numéro de la parcelle dans laquelle la parcelle j a été fusionnée.

Sur l'exemple, on obtient le vecteur

$$\boldsymbol{parent} = \begin{pmatrix} 11 & 11 & 13 & 13 & 15 & 17 & 14 & 14 & 12 & 12 & 18 & 16 & 15 & 16 & 17 & 19 & 18 & 19 & 19 \end{pmatrix}$$

Pour obtenir la parcellisation, on trouve d'abord à l'aide de la matrice $\boldsymbol{mergedClusters}$ les indices des parcelles gardées. À l'aide du vecteur $\boldsymbol{parent}$, on répartit chaque élément $\boldsymbol{x}_i$ dans les

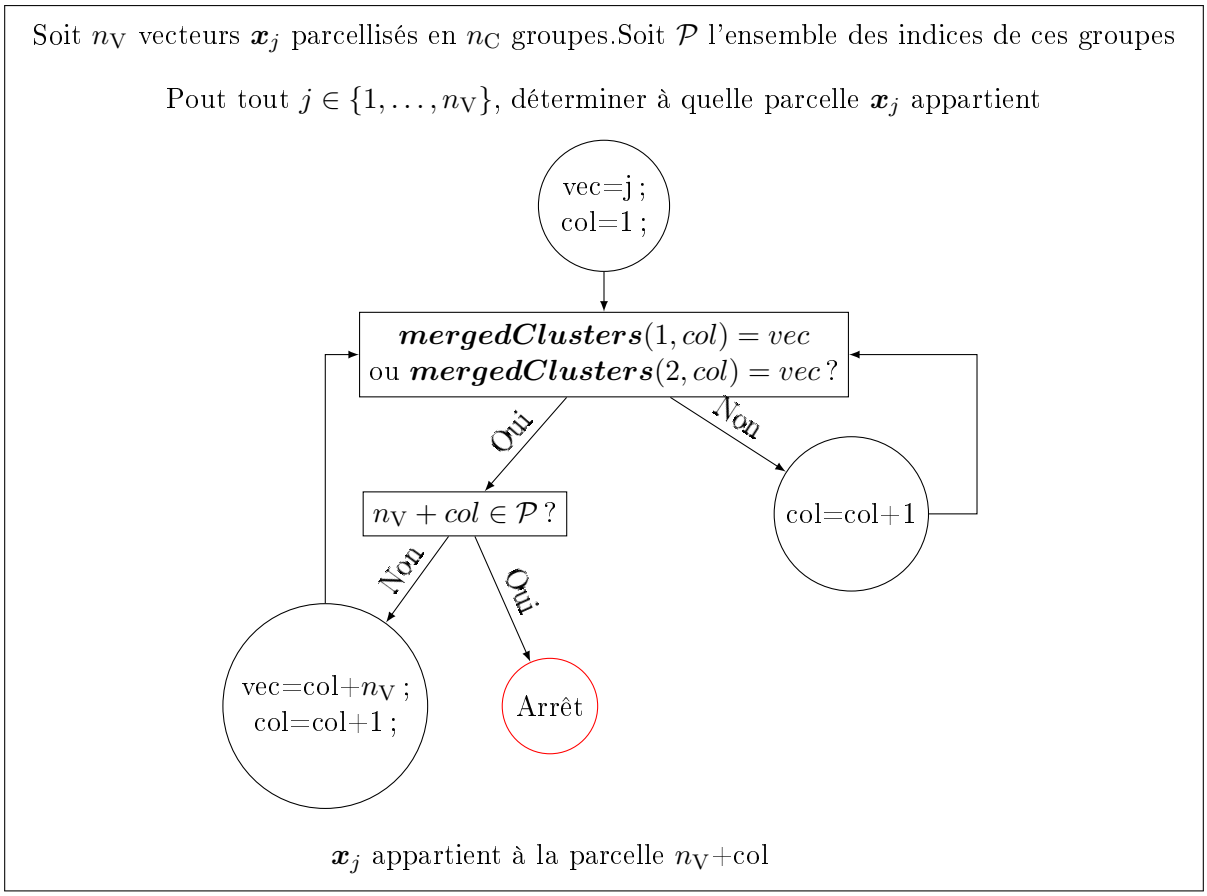


FIGURE 6.27 – Obtention de la composition des clusters à partir de ***mergedClusters***.

parcelles gardées. Pour cela, il faut calculer pour chaque vecteur $\boldsymbol{x}_j$ $p = \boldsymbol{parent}(j)$ et si p n'est pas l'indice d'une des parcelles gardées, on calcule $p = \boldsymbol{parent}(p)$ récursivement jusqu'à tomber sur une parcelle p sélectionnée. La figure 6.28 illustre le processus récursif. Évaluons la complexité de ce nouveau processus. Si le vecteur $\boldsymbol{x}_j$ est une parcelle sélectionnée, alors le processus s'arrête en ayant fait n_C comparaisons pour tester si $\boldsymbol{x}_j$ fait partie des n_C parcelles sélectionnées. Sinon, on effectue n_C comparaisons et l'appel à la fonction avec $p = \boldsymbol{parent}(p)$. Si on note $T(n)$ le nombre d'opérations élémentaires effectuées à l'étape n, on a $T(n) = T(n-1) + n_C$. On peut montrer que la complexité du processus est en $O(n_V)$ [46]. Donc la complexité pour traiter tous les vecteurs est en en $O(n_V^2)$. La complexité a diminué d'un ordre de grandeur par rapport au cas où on n'utilise que ***mergedClusters***.

6.2.3.2 Comment trouver la distance minimale et mettre à jour la distance ?

Il faut stocker la matrice de distances entre les vecteurs initiaux puis la distance entre les parcelles. Remarquons déjà que pour les parcelles initiales, par symétrie de la distance et puisque la distance d'une parcelle avec elle-même n'intervient pas dans l'algorithme (puisqu'on ne regroupe pas la parcelle avec elle-même), on peut ne stocker que la partie triangulaire supérieure stricte de la matrice de distances. Reste à savoir où stocker à chaque itération la distance entre la nouvelle parcelle fusionnée et les autres parcelles. Puisque l'ultime parcelle (contenant tous les éléments) a pour indice $2n_V - 1$, on pourrait penser stocker une matrice de distances de taille $(2n_V - 1)(2n_V - 2)/2$.Cette solution est à exclure si l'algorithme doit traiter un grand nombre de données pour limiter la consommation mémoire. La distance de la parcelle $C_{i \cup j}$ issue de la fusion de la parcelle C_i et de la parcelle C_j avec les autres parcelles sera stockée à la place de la distance entre la parcelle $C_{\min(i,j)}$ et les autres parcelles. Autrement dit, la parcelle $C_{i \cup j}^{k+1}$ issue

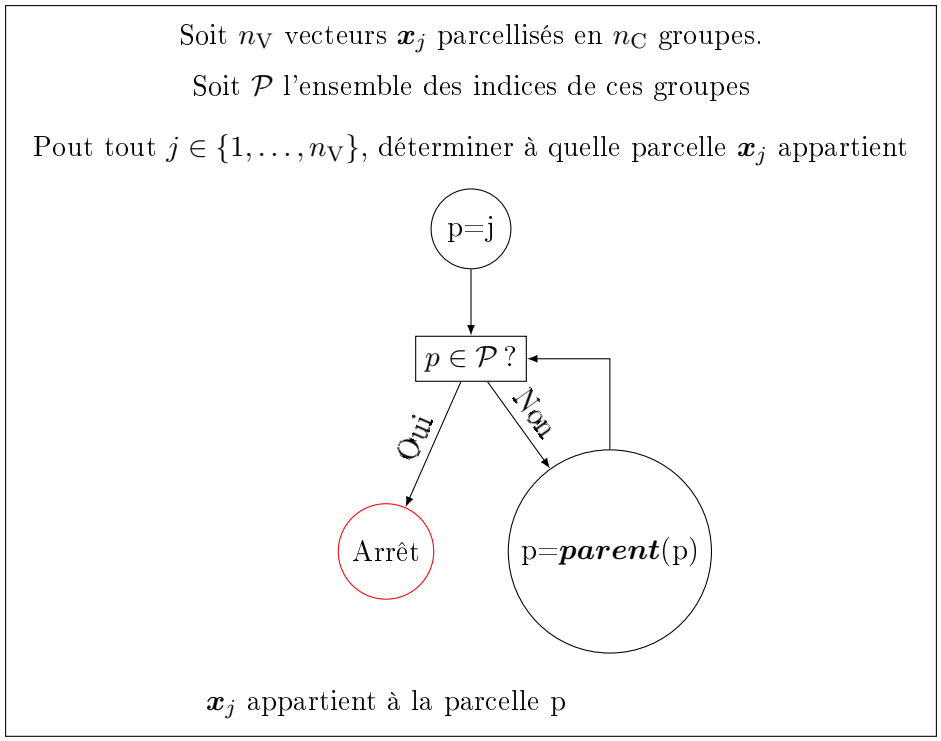


FIGURE 6.28 – Obtention de la composition des clusters à partir de ***parent***.

de la fusion de la parcelle C_i^k et de la parcelle C_j^k aura pour indice $n_V + k$ mais sa distance sera stockée dans la colonne $\min(i, j)$.

La formule 6.10 démontrée en annexe B fait intervenir, en plus des distances, la taille des parcelles. Il faut donc garder la taille des parcelles. Comme pour la distance, la taille de la parcelle $C_{i\cup j}$ issue de la fusion de la parcelle C_i et de la parcelle C_j est stockée à l'indice $\min(i, j)$ et la taille de la parcelle d'indice $\max(i, j)$ vaudra alors 0 puisque cette parcelle n'existe plus. Elle permet de ne pas calculer le centroïde de la parcelle nouvellement formée et utilise uniquement des distances déjà calculées.

À chaque itération, il faut trouver quelle paire de parcelles est séparée par la distance minimale. Une possibilité est de parcourir à chaque itération la matrice de distances pour trouver son minimum. Cependant l'opération est coûteuse puisque dans le pire cas, il faut parcourir toute la matrice qui est de taille $(n_V(n_V - 1)/2$.Pour y remédier, on utilise deux vecteurs de taille n_V : un vecteur ***minDist*** qui contient pour chaque parcelle sa distance minimale avec les autres parcelles et un vecteur ***minLoc*** contenant l'indice de la parcelle avec laquelle la distance minimale est atteinte. L'idée est qu'il suffit alors de chercher le minimum dans le premier vecteur qui est de taille n_V au lieu de $(n_V(n_V - 1)/2$.Toutefois, il faut mettre à jour ce vecteur à chaque itération. À chaque itération, peut se produire deux cas : soit la distance minimale est atteinte pour une des parcelles fusionnées et dans ce cas, il faut recalculer la distance minimale ; soit la distance minimale n'est pas atteinte pour une des parcelles fusionnées et il suffit alors de comparer la distance minimale avec la distance entre la parcelle considérée et la nouvelle parcelle fusionnée.. L'idée est que pour les premières itérations où le nombre de parcelles est grand, ce sera la deuxième situation qui se produira et donc l'étape de mise à jour sera moins coûteuse. L'algorithme 6 montre comment mettre à jour ces deux vecteurs à chaque itération.

6.2.4 Analyse de complexité

6.2.4.1 Complexité en espace mémoire

Théorème 6.1. *La complexité en espace mémoire est en* $O(n_V^2)$.

Algorithme 6 : Étape de mise à jour de ***minDist*** et ***minLoc***.

Entrée : Si on note $i^* = \arg\min \boldsymbol{minDist}$ et $j^* = \boldsymbol{minLoc}(i^*)$ les indices des deux parcelles fusionnées, on a upInd=min(i^*, j^*) la parcelle gardée et delInd=max(i^*, j^*) la parcelle devenant inactive

```
1  minDist(delInd,1)=∞
2  pour j de 1 à n_V faire
       // ne considérer que les clusters actifs
3      si dimClusters(j)≠0 alors
           // Recalculer minDist si minLoc(j) est dans le cluster delInd ou
              upInd
4          si (minLoc(j)=delInd) ou (minLoc(j)=upInd) alors
5              minDist(j)← ∞
6              pour i de 1 à j-1 faire
7                  si dimClusters(i) ≠0 alors
8                      d=D(i,j)
9                      si (d<minDist(j)) ou (d=minDist(j) et i<minLoc(j)) alors
10                         minDist(j)← d
11                         minLoc(j)←i
12             pour i de j+1 à n_V faire
13                 si dimClusters(i)≠0 alors
14                     d=D(i,j)
15                     si d<minDist(j) ou (d=minDist(j) et i<minLoc(j)) alors
16                         minDist(j)← d
17                         minLoc(j)←i
18         sinon
               // Calculer la distance avec le nouveau cluster formé
19             d=D(j,upDim)
20             si d<minDist(j) ou (d=minDist(j) et upInd<minLoc(j)) alors
21                 minDist(j)← d
22                 minLoc(j)←upInd
```

Démonstration du théorème 6.1. Considérons les variables nécessaires pour l'algorithme :

- ***X*** de taille $n_\mathrm{T} \times n_\mathrm{V}$: la matrice de données
- ***mergedClusters*** de taille $2 \times (n_\mathrm{V} - 1)$
- ***parent*** de taille $2n_\mathrm{V} - 1$
- ***D*** vecteur de taille $\frac{n_\mathrm{V}(n_\mathrm{V}-1)}{2}$ contenant la distance
- ***minDist*** vecteur de taille n_V contenant pour chaque parcelle sa distance minimale avec les autres parcelles
- ***minLoc*** vecteur de taille n_V contenant pour chaque parcelle i avec quelle parcelle $j \neq i$ la distance minimale est atteinte
- ***dimClusters*** vecteur de taille n_V contenant le cardinal des parcelles

Nous avons quatre vecteur en $\mathrm{O}(n_\mathrm{V})$, une matrice en $\mathrm{O}(n_\mathrm{V})$ et deux matrices en $\mathrm{O}(n_\mathrm{V}^2)$. La complexité en espace mémoire est donc en $\mathrm{O}(n_\mathrm{V}^2)$. □

6.2.4.2 Complexité en temps

Théorème 6.2. *La complexité du calcul de la distance euclidienne est en* $O(n_\mathrm{V}^2 n_\mathrm{T})$

Preuve du théorème 6.2. on calcule les distances euclidiennes entre chaque paire d'éléments initiaux. Or, on a $\binom{n_V}{2} = n_V(n_V-1)/2$ paires d'éléments initiaux. Intéressons-nous maintenant à la distance euclidienne : pour chaque composante des vecteurs, on effectue deux opérations : une soustraction et une multiplication (pour l'élévation au carré) puis les résultats des composantes sont sommés. On effectue donc $3n_T-1$ opérations pour calculer la distance euclidienne. Le calcul de la distance euclidienne a donc une complexité en $O(n_T)$. Comme ce calcul est répété pour chaque paire, la complexité pour le calcul de la distance est $O(n_V^2 n_T)$. □

Théorème 6.3. *Dans le pire des cas, les itérations se font en* $O\left(\frac{n_V^3}{3}\right)$.

Preuve du théorème 6.3. On a vu en section 6.1.1 qu'il faut $n_V - 1$ itérations pour obtenir une seule parcelle. Pour déterminer la complexité du calcul des itérations, détaillons ce qui se passe pour une itération :

1. Calcul de l'indice du minimum de ***minDist***
2. Mise à jour des distances pour la parcelle supprimée et pour la nouvelle parcelle
3. Mise à jour de ***minDist*** et ***minLoc***
4. Remplissage de ***mergedCluster*** et ***parent***

La première étape est en $O(n_V)$ (**cf. Lemme 6.1**). La deuxième étape est en $O(2n_V - k)$ (**cf. Lemme 6.2**). La troisième étape est dans le pire des cas en $O(n_V - k)^2$ (**cf. Lemme 6.3**). La quatrième étape est en $O(n_V)$ (**cf. Lemme 6.4**). On répète les quatre étapes à chaque itération. Le complexité des itérations est donc donnée par :

$$\sum_{k=1}^{n_V-1} (n_V + n_V + 2n_V - k + (n_V - k)^2 + n_V) \ . \tag{6.11}$$

Or, on a :

$$\sum_{k=1}^{n_V-1} k = \frac{n_V(n_V-1)}{2} \ .$$

De plus, en posant $K = n_V - k$, on obtient

$$\begin{aligned}\sum_{k=1}^{n_V-1} k^2 &= \sum_{K=1}^{n_V-1} K^2 \\ &= \frac{(n_V-1)(n_V-2)(2n_V-1)}{6}\end{aligned}$$

En revenant à l'équation (6.11), les itérations se font donc dans le pire des cas en $O\left(\frac{n_V^3}{3}\right)$. □

Lemme 6.1. *Trouver l'indice de la distance minimale se fait en* $O(n_V)$ *opérations.*

Preuve du lemme 6.1. Pour trouver l'indice de la distance minimale, il faut parcourir ***minDist*** pour trouver le minimum. Or ***minDist*** est de taille n_V (qui représente le nombre initial de parcelles). Il faut donc $O(n_V)$ opérations. □

Lemme 6.2. *L'étape de mise à jour du vecteur de distance à l'itération* k *s'effectue en* $O(2n_V - k)$ *opérations.*

Preuve du lemme 6.2. La mise à jour du vecteur de distance se fait en deux étapes. La première consiste à mettre à l'infini la distance entre la parcelle qui n'est plus active et les autres parcelles. La mise à jour s'effectue donc pour les $n_V - 1$ paires : elle est donc en $O(n_V)$.

La deuxième étape met à jour la distance entre la nouvelle parcelle et les autres parcelles encore actives. Comme on fusionne les parcelles deux à deux à chaque itération, à l'itération

k, il y a $n_V - k$ parcelles actives. La deuxième sous-étape s'effectue donc en $n_V - k$ mises à jour. Or, la nouvelle distance est calculée par la formule de Lance-Williams et nécessite douze opérations élémentaires (six additions, une soustraction, quatre multiplications et une division). La deuxième étape s'effectue donc en $O(n_V - k)$ opérations. □

Lemme 6.3. *La mise à jour du vecteur* ***minDist*** *à l'itération k se fait en $O((n_V - k)^2)$ opérations.*

Preuve du lemme 6.3. On cherche pour chaque parcelle active, la distance minimale entre cette parcelle et les autres parcelles actives. Plaçons-nous à l'itération k et considérons une parcelle C_l^k encore active mais qui n'est pas fusionnée à cette itération. Si la distance minimale entre la parcelle C_l^k et les parcelles actives à l'itération $k-1$ est atteinte pour une parcelle qui n'est pas fusionnée à l'itération k, il suffit de comparer cette distance avec la distance entre la parcelle C_l^k et la nouvelle parcelle à l'itération k.

En revanche, si cette distance minimale est atteinte pour une des parcelles fusionnées, il faut recalculer la distance minimale en parcourant les distances entre la parcelle l et les autres parcelles encore actives, ce qui se fait en $O(n_V - k)$ comparaisons. Pour la parcelle nouvellement formée, il faut recalculer la distance minimale. Dans le pire des cas, il faut recalculer la distance minimale pour chacune des parcelles encore actives donc la troisième étape s'effectue dans le pire des cas en $(n_V - k)^2$ calculs. □

Lemme 6.4. *Le remplissage de* ***parent*** *et* ***mergedClusters*** *est en $O(n_V)$.*

Preuve du lemme 6.4. Pour remplir ***parent***, on cherche pour les indices des deux parcelles qui sont fusionnées la parcelle parente. En effet, afin de gagner de la place mémoire pour les tableaux utilisés dans l'algorithme, lorsqu'on fusionne la parcelle i et la parcelle j, la parcelle $i \cup j$ est placée à l'indice $\min(i, j)$. Cependant, dans les tableaux ***mergedClusters*** et ***parent***, on a besoin d'un nouvel indice pour représenter les parcelles fusionnées. La parcelle $i \cup j$ correspond alors à la parcelle $n_V + k$ où k est l'indice de l'itération où les parcelles i et j ont été fusionnées. Dans le pire des cas, on parcourt tout le vecteur ***parent***. L'étape se fait donc en $O(n_V)$.

Le remplissage de ***mergedClusters*** nécessite deux affectations donc et en $O(1)$. □

Théorème 6.4. *L'algorithme de Ward s'effectue dans le pire des cas en $O\left(\frac{n_V^2 n_T}{2} + \frac{n_V^3}{3}\right)$.*

Preuve du théorème 6.4. L'algorithme de Ward est décomposé en deux étapes : le calcul de la distance euclidienne pour chaque paire de vecteurs qui s'effectue en $O(n_T n_V^2)$ (**cf. Théorème 6.2**) et les itérations qui s'effectuent dans le pire des cas en $O\left(\frac{n_V^3}{3}\right)$ (**cf. Théorème 6.3**). Donc, l'algorithme de Ward se fait dans le pire des cas en $O\left(\frac{n_V^2 n_T}{2} + \frac{n_V^3}{3}\right)$. □

On constate que la complexité des itérations est plus grande que la complexité du calcul de la distance, ce qui est en contradiction par rapport à la répartition du temps de calcul montré à la table 6.2. Cependant, la complexité de la distance est la complexité réelle alors que la complexité des itérations est donnée dans le pire cas. Or cette complexité dans le pire cas ne se produit que lorsqu'à chaque itération les parcelles sont séparées par exactement la même distance, ce qui donne une chaîne sur le dendrogramme. Il est donc peu probable que ce cas arrive. Malgré une complexité théorique plus élevée, l'exécution des itérations est plus rapide que le calcul de la distance.

6.3 Parallélisation de l'algorithme de Ward

6.3.1 Pourquoi paralléliser ?

La section 6.2.4 montre que le nombre d'opérations de l'algorithme de Ward est en $O(n_V^2 n_T + n_V^3)$. Rappelons maintenant que l'objectif de la thèse est de traiter des données d'IRMf. Pour ces

données d'IRMf, chaque colonne de la matrice $\boldsymbol{X}$ correspond au signal temporel mesuré dans un voxel puisque le but est de regrouper les voxels qui ont une réponse temporelle similaire. Prenons le cas d'un sujet, on a $n_{\mathrm{T}} \in \{100, \ldots, 1000\}$ et $n_{\mathrm{V}} \in \{10^4, \ldots, 5 \times 10^5\}$.

Regardons le temps de calcul estimé pour les bornes supérieures ($n_{\mathrm{T}} = 1000$ et $n_{\mathrm{V}} = 5 \times 10^5$). Le nombre d'opérations est estimé à environ 2.5×10^{14}. Supposons que le CPU est à 1 GFlops (c'est-à-dire qu'il effectue 10^9 opération flottante par seconde), le temps théorique est d'environ 3 jours. Et cela, rien que pour calculer la distance entre les voxels.

Pour pouvoir traiter de grands jeux de données, il faut donc trouver des stratégies de parallélisation pour réduire le temps de calcul.

Nous avons envisagé trois parallélisations possibles : une première parallélisation uniquement avec OpenMP à la fois pour la distance et les mises à jour de la distance à chaque itération ; une deuxième parallélisation utilisant le GPU pour le calcul de la distance et une troisième parallélisation utilisant OpenMP et le GPU.

6.3.2 Les différentes versions de la parallélisation

Nous proposons plusieurs versions parallélisées de l'algorithme de Ward, qui s'appuient principalement sur les différentes parallélisations du calcul de la distance euclidienne présentées au chapitre 7. En effet, en reprenant l'algorithme de Ward, on distingue deux parties à paralléliser : la distance et les mises à jour. La figure 6.29 montre les différentes parties de l'algorithme qui sont parallélisées : les cadres rouges montrant la parallélisation avec OpenMP et les cadres bleus la parallélisation avec CUDA. Seul le calcul de la distance a été parallélisé avec CUDA. L'étape de mise à jour des distances est difficilement portable sur GPU. Bien qu'on puisse copier sur GPU les parties du vecteur de distance correspondant aux distances entre les deux parcelles fusionnées et les autres parcelles, la mise à jour de la distance ne se fait que si la parcelle est encore active. Il faut donc tester si la parcelle est encore active par un branchement. Les processus du GPU n'exécuteront pas le même code si la parcelle est active ou inactive. L'exécution sur GPU se fera donc en séquentiel et donc on perdrait le bénéfice d'une implémentation sur GPU. La table 6.3 récapitule les différentes parallélisations développées avec la référence aux sections décrivant la parallélisation. La deuxième version de la parallélisation apporte essentiellement des améliorations pour le calcul de la distance. La parallélisation des étapes de mise à jour de la distance est décrite dans la section 6.3.3

Approches	Calcul de la distance	Mise à jour des distances et de ***minDist***
$1^{ère}$ approche	OpenMP (**cf. 7.3.1**)	OpenMP(**cf 6.3.3**)
$2^{ème}$ approche	CUDA (**cf. 7.3.2**)	Séquentiel
$3^{ème}$ approche	CUDA (**cf. 7.3.2**)	OpenMP (**cf. 6.3.3**)

(a) $1^{ère}$ version

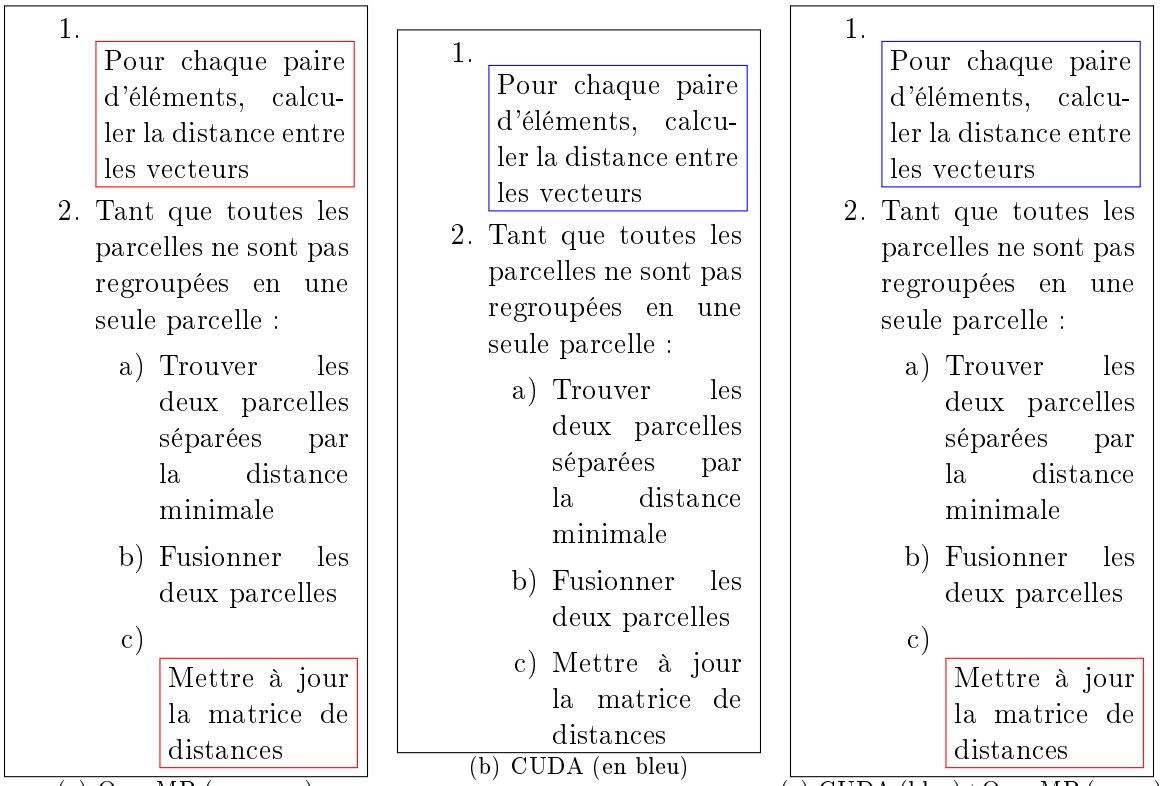
Approche	Calcul de la distance	Mise à jour des distances et de ***minDist***
$1^{ère}$ approche	OpenMP (**cf. 7.4.1**)	OpenMP**cf. 6.3.3**
$2^{ème}$ approche	CUDA (**cf. 7.4.2**)	Séquentiel
$3^{ème}$ approche	CUDA+OpenMP (**cf. 7.4.3**)	OpenMP (**cf. 6.3.3**)

(b) $2^{ème}$ version

Approche	Calcul de la distance	Mise à jour des distances et de ***minDist***
$1^{ère}$ approche	OpenMP (**cf. 7.5.2.1**)	OpenMP (**cf. 6.3.3**)
$2^{ème}$ approche	CUDA (**cf. 7.5.2.2**)	Séquentiel

(c) $3^{ème}$ version

TABLE 6.3 – Synthèse des trois approches parallèles



(a) OpenMP (en rouge) (b) CUDA (en bleu) (c) CUDA (bleu)+OpenMP (rouge)

FIGURE 6.29 – Les différentes parallélisations possibles de l'algorithme de Ward.

6.3.3 Parallélisation de l'étape de mise à jour des distances avec OpenMP

L'algorithme de Ward étant itératif (fusion des parcelles deux à deux), on ne peut paralléliser que des sous-étapes à chaque itération. En effet, les étapes de mise à jour des distances et des vecteurs ***minDist*** et ***minLoc*** peuvent se faire séparément pour chaque parcelle. Ces mises à jour sont donc distribuées sur les différents processus légers.

Pour cela, on ouvre une session de processeurs parallèles avant le calcul des distances, ce qui permet en plus d'éviter de recréer à chaque itération une session parallèle ce qui est coûteux. Cela oblige alors à utiliser des sections OMP SINGLE pour calculer quelles parcelles fusionner et remplir les vecteurs de sortie afin que chaque processus léger ait les bonnes valeurs en mémoire. Une des difficultés notamment pour le calcul des distances est d'éviter le *false sharing* de la mémoire.

Ce chapitre décrit l'algorithme de Ward, son implémentation et sa parallélisation. Les résultats numériques de la parallélisation sont donnés en section 8.3. Le chapitre suivant est consacré au calcul de la distance euclidienne.

Chapitre 7

Calcul de la distance euclidienne

Le chapitre 6 a montré que le calcul de la distance représente une part importante du temps d'exécution de l'algorithme de Ward. Une grande partie de la thèse a donc porté sur l'amélioration du calcul de distance et sa parallélisation efficace. Ces efforts sont d'autant plus justifiés que le calcul de la distance peut servir dans d'autres applications que l'algorithme de Ward. La première partie explore les différentes façons de calculer la distance en séquentiel pour paralléliser par la suite un algorithme proche de l'optimal. La deuxième partie explique comment passer d'une matrice de distances à un vecteur de distances. La troisième partie présente la première version de la parallélisation effectuée. La quatrième partie décrit la deuxième version de la parallélisation et la cinquième partie décrit la dernière approche développée qui est *cache-aware*.

7.1 Calcul de la distance en séquentiel

Il existe différentes manières de calculer la matrice de distances. Sont présentés ci-dessous différents algorithmes allant de l'algorithme classique par double boucle aux algorithmes par découpage en tuiles en passant par une version vectorielle et une version matricielle de la distance. La manière la plus naturelle pour calculer la matrice de distances est décrite dans l'algorithme 7 et consiste à parcourir les éléments avec une double boucle (**cf. lignes 1 et 2**).

Algorithme 7 : Calcul des distances par double boucle.

Entrée : $\boldsymbol{X}$ matrice de taille $n_\mathrm{T} \times n_\mathrm{V}$
Sortie : $\boldsymbol{D}$ matrice de taille $n_\mathrm{V} \times n_\mathrm{V}$

```
1 pour j de 1 à n_V faire
2     pour i de j+1 à n_V faire
3         D(i, j)=∑_{t=1}^{n_T}(x_tj − x_ti)^2
```

On peut toutefois vectoriser une partie des opérations. Posons $\boldsymbol{A} = \boldsymbol{x}_j \cdot \mathbf{1}_{\boldsymbol{n_\mathbf{V}}}^\top$, on a alors

$$\boldsymbol{D}_j = \left(\sum^{col} (\boldsymbol{A} - \boldsymbol{X})^{\hat{}2} \right)^\top \quad .$$

Cette formule a plusieurs implémentations pour le calcul de $\boldsymbol{A}$: soit par multiplication matricielle soit en répétant la colonne $\boldsymbol{x}_j$. L'algorithme 8 montre comment utiliser cette formule : on n'utilise plus qu'une seule boucle (en ligne 1) en remplaçant la boucle interne par des opérations vectorielles en ligne 3.

On peut pousser la vectorisation plus loin en utilisant uniquement des matrices. Posons pour cela

$$\boldsymbol{S} = \sum^{col} \boldsymbol{X}^{\hat{}2} \cdot \mathbf{1}_{\boldsymbol{n_\mathbf{V}}} \quad ,$$

Algorithme 8 : Calcul des distances sous forme vectorielle.

Entrée : $\boldsymbol{X}$ matrice de taille $n_\mathrm{T} \times n_\mathrm{V}$
Sortie : $\boldsymbol{D}$ matrice de taille $n_\mathrm{V} \times n_\mathrm{V}$

1 **pour** *j de 1 à* n_V **faire**
2 $\quad \boldsymbol{A}_j = \boldsymbol{x}_j \cdot \mathbf{1}_{\boldsymbol{n_\mathrm{V}}}^\top$
3 $\quad \boldsymbol{D}_j \leftarrow \sum^{col} (\boldsymbol{A} - \boldsymbol{X})^{\hat{}2}$

on peut alors calculer $\boldsymbol{D}$ par $\boldsymbol{D} = \boldsymbol{S} + \boldsymbol{S}^\top - 2\boldsymbol{X}^\top \cdot \boldsymbol{X}$. L'algorithme 9 montre comment utiliser cette formule : les deux boucles « Pour » (lignes 1 et 2 de l'algorithme 7) ont disparu au profit d'opérations matricielles en lignes 1 et 2.

Algorithme 9 : Calcul des distances sous forme matricielle.

Entrée : $\boldsymbol{X}$ matrice de taille $n_\mathrm{T} \times n_\mathrm{V}$
Sortie : $\boldsymbol{D}$ matrice de taille $n_\mathrm{V} \times n_\mathrm{V}$

1 $\boldsymbol{S} = \sum^{col} \boldsymbol{X}^{\hat{}2} \cdot \mathbf{1}_{\boldsymbol{n_\mathrm{V}}}$
2 $\boldsymbol{D} = \boldsymbol{S} + \boldsymbol{S}^\top - 2\boldsymbol{X}^\top \cdot \boldsymbol{X}$

Un dernier algorithme est envisagé : l'idée est de reprendre l'algorithme avec la double boucle mais en exploitant mieux la localité des données. Pour cela, on propose de découper la matrice de données en tuiles rectangulaires : la figure 7.1 et l'annexe C illustre l'algorithme qui est donné par l'algorithme 10. La double boucle sur les colonnes est remplacée par une double boucle sur les tuiles en lignes 2 et 7. La ligne 3 calcule les indices de colonnes et de fin de la tuile d'indice K. Le minimum permet de tenir compte du cas où n_V n'est pas un multiple de m_C. À l'intérieur de la boucle 2, on calcule la distance entre chaque paire de colonnes de la tuile considérée (**cf. lignes 4 et 5**). Puis à l'intérieur de la double boucle, on calcule la distance entre les paires de colonnes de deux tuiles différentes (**cf. lignes 9 et 10**). La ligne 8 calcule les indices de colonnes et de fin de la tuile d'indice L.

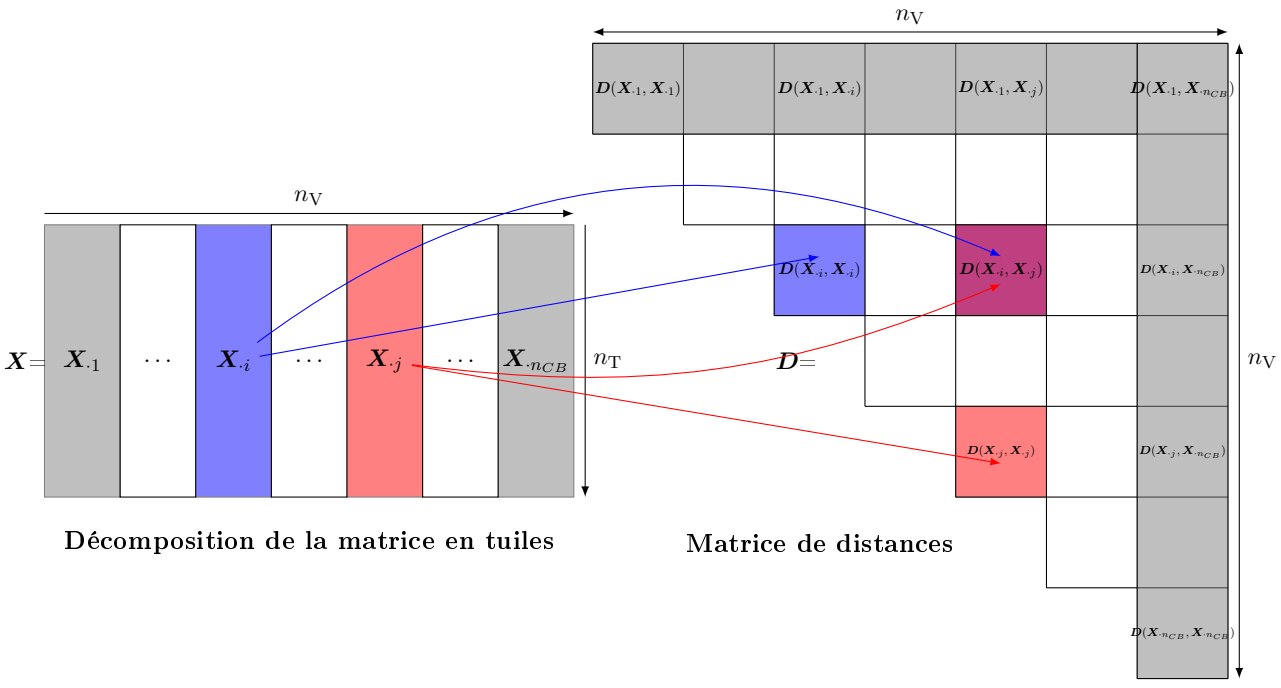


FIGURE 7.1 – Calcul de la distance par découpage en tuiles.

Algorithme 10 : Calcul de distance par tuile rectangulaire.

Entrée : $\boldsymbol{X}$ une matrice de taille $n_\mathrm{T} \times n_\mathrm{V}$
m_C la taille de la tuile
Sortie : $\boldsymbol{D}$ matrice de taille $n_\mathrm{V} \times n_\mathrm{V}$

```
// Calcul du nombre de tuiles pour couvrir la matrice
```
1 $n_{CB} = \left\lceil \frac{n_\mathrm{V}}{m_C} \right\rceil$
```
// Parcours des tuiles en colonnes
```
2 **pour** *K de 1 à* n_{CB} **faire**
3 KBeg=1+(K-1)$\times m_C$;KEnd=min$(K \times m_C, n_\mathrm{V})$
```
   // Calcul des distances entre les colonnes de la tuile X.K
```
4 **pour** *j de KBeg à KEnd* **faire**
5 **pour** *i de KBeg à KEnd* **faire**
6 $\boldsymbol{D}(i,j) = \sum_{t=1}^{n_\mathrm{T}} (x_{ti} - x_{tj})^2$
```
   // Calcul de la distance entre colonnes de la tuile X.K avec les
      colonnes de la tuile X.L
```
7 **pour** *L de K+1 à* n_{CB} **faire**
8 LBeg=1+(L-1)$\times m_C$;LEnd=min$(L \times m_C, n_\mathrm{V})$
9 **pour** *j de KBeg à KEnd* **faire**
10 **pour** *i de LBeg à LEnd* **faire**
11 $\boldsymbol{D}(i,j) = \sum_{t=1}^{n_\mathrm{T}} (x_{ti} - x_{tj})^2$
12 $\boldsymbol{D}(j,i) = \boldsymbol{D}(i,j)$

Une dernière amélioration est de combiner le découpage en tuiles au calcul matriciel de la distance pour les sous-matrices de la matrice de distances présenté à l'algorithme 11. Les doubles boucles de l'algorithme 10 en lignes 4 et 5 et en lignes 9 et 10 sont remplacées par deux opérations matricielles en lignes 3 et 4, puis en lignes 6 et 7. Dans les formules, on note $\boldsymbol{X}_{.K}$ la tuile couvrant les colonnes KBeg à KEnd et $\boldsymbol{X}_{.L}$ la tuile couvrant les colonnes LBeg à LEnd.

Algorithme 11 : Calcul de distance par tuile rectangulaire avec opérations matricielles.

Entrée : $\boldsymbol{X}$ matrice de taille $n_\mathrm{T} \times n_\mathrm{V}$
m_C taille de la tuile
Sortie : $\boldsymbol{D}$ matrice de taille $n_\mathrm{V} \times n_\mathrm{V}$

```
// Nombre de tuiles pour couvrir toute la matrice
```
1 $n_{CB} = \left\lceil \frac{n_\mathrm{V}}{m_C} \right\rceil$
```
// Parcours des tuiles en colonnes
```
2 **pour** *K de 1 à* n_{CB} **faire**
```
   // Calcul des distances entre les colonnes de la tuile X.K
```
3 $\boldsymbol{S}_1 = \sum^{col} \boldsymbol{X}_{.K}{}^{\hat{}2} \cdot \mathbf{1}_{\boldsymbol{n_\mathrm{V}}}$
4 $\boldsymbol{D}(\cdot K, \cdot K) = \boldsymbol{S}_1 + \boldsymbol{S}_1^\top - 2\boldsymbol{X}_{.K}^\top \cdot \boldsymbol{X}_{.K}$
```
   // Calcul de la distance entre colonnes de la tuile X.K avec les
      colonnes de la tuile X.L
```
5 **pour** *L de K+1 à* n_{CB} **faire**
6 $\boldsymbol{S}_2 = \sum^{col} \boldsymbol{X}_{.L}{}^{\hat{}2} \cdot \mathbf{1}_{\boldsymbol{n_\mathrm{V}}}$
7 $\boldsymbol{D}_{KL} = \boldsymbol{S}_2 + \boldsymbol{S}_1^\top - 2\boldsymbol{X}_{.K}^\top \times \boldsymbol{X}_{.L}$

n_V	Double précision	Simple précision
10 000	0,8	0,4
30 000	7,2	3,6
50 000	20	10
70 000	39,2	19,6
100 000	80	40
120 000	115,2	57,6

TABLE 7.1 – Taille de la matrice de distances de taille $n_\mathrm{V} \times n_\mathrm{V}$ en Go.

7.2 Nombres triangulaires et conversion d'indices vectoriels/matriciels

Maintenant que nous avons établi un algorithme efficace pour le calcul de la distance, il faut s'intéresser au stockage de la matrice de distances. La table 7.1 montre la taille de la matrice de distances en fonction du nombre de colonnes n_V de la matrice $\boldsymbol{X}$. On voit bien que garder la distance sous forme matricielle devient rapidement prohibitif puisque sur des machines de calcul de bureau en double précision pour $n_\mathrm{V} = 120\,000$, la matrice de distances ne peut pas être stockée en mémoire RAM.

Cependant, la matrice de distances est symétrique donc on peut ne garder qu'une partie triangulaire de la matrice de distances. De plus, la distance d'un vecteur avec lui-même est nulle donc on peut également ne pas stocker ni calculer la diagonale de la matrice. On propose donc de garder uniquement la partie triangulaire (stricte ou non selon les cas) et de la représenter sous forme de tableau. Pour cela, on propose de ranger les cases des matrices triangulaires en les lisant colonne par colonne. Le problème est qu'en passant sous forme vectorielle, la distance entre $\boldsymbol{x}_i$ et $\boldsymbol{x}_j$ n'est plus rangée dans la case $\boldsymbol{D}(i,j)$ ou la case $\boldsymbol{D}(j,i)$ mais dans la case k du vecteur de distances.

Les sous-sections suivantes établissent la transformation permettant de passer des indices matriciels à l'indice vectoriel et la transformation inverse permettant d'obtenir les indices matriciels à partir de l'indice vectoriel. Les figures 7.2 et 7.3 illustrent le passage des indices matriciels vers l'indice vectoriel, même si pour plus de clarté, le vecteur est représenté sous forme matricielle. On remarque que dans le cas d'une matrice triangulaire supérieure stricte (**cf. Figure 7.3**), on a un décalage des diagonales par rapport à une matrice triangulaire supérieure pour obtenir les indices vectoriels k. La figure 7.4 montre le passage des indices matriciels à l'indice vectoriel pour une matrice triangulaire inférieure où les cases sont stockées colonnes par colonnes. Les sections 7.2.2, 7.2.3 traitent les transformations des matrices triangulaires supérieures strictes et non strictes vers un vecteur et les sections 7.2.5 et 7.2.4 traitent les transformations pour les matrices triangulaires inférieures.

On cherche à établir l'expression de la fonction ϕ qui associe à un couple d'indices matriciels (i,j) son indice vectoriel k :

$$\phi\colon \begin{cases} \mathbb{N}^2 \longrightarrow \mathbb{N} \\ (i,j) \longmapsto k \end{cases},$$

et l'expression de sa fonction réciproque $\psi = \phi^{-1}$ qui à un indice vectoriel k associe le couple d'indices matriciels (i,j) :

$$\psi\colon \begin{cases} \mathbb{N} \longrightarrow \mathbb{N}^2 \\ k \longmapsto (i,j) \end{cases},$$

Pour distinguer les trois cas présentés en figures 7.2, 7.3 et 7.4, on note par la suite ϕ_1 et ψ_1 les transformations pour le cas des matrices triangulaires supérieures, ϕ_2 et ψ_2 les transformations pour le cas des matrices triangulaires supérieures strictes, ϕ_3 et ψ_3 les transformations pour le

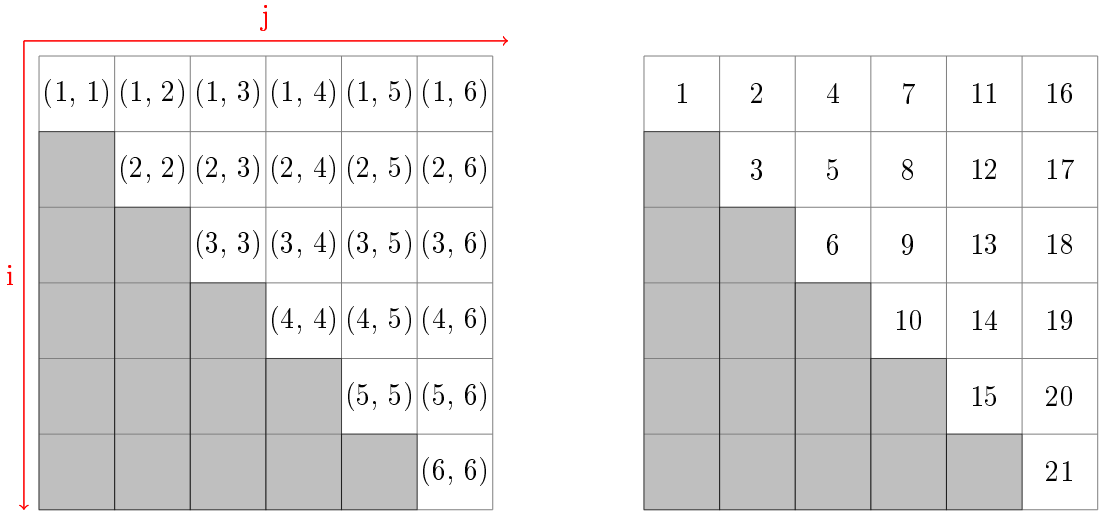


FIGURE 7.2 – Transformation d'une matrice triangulaire supérieure en vecteur.

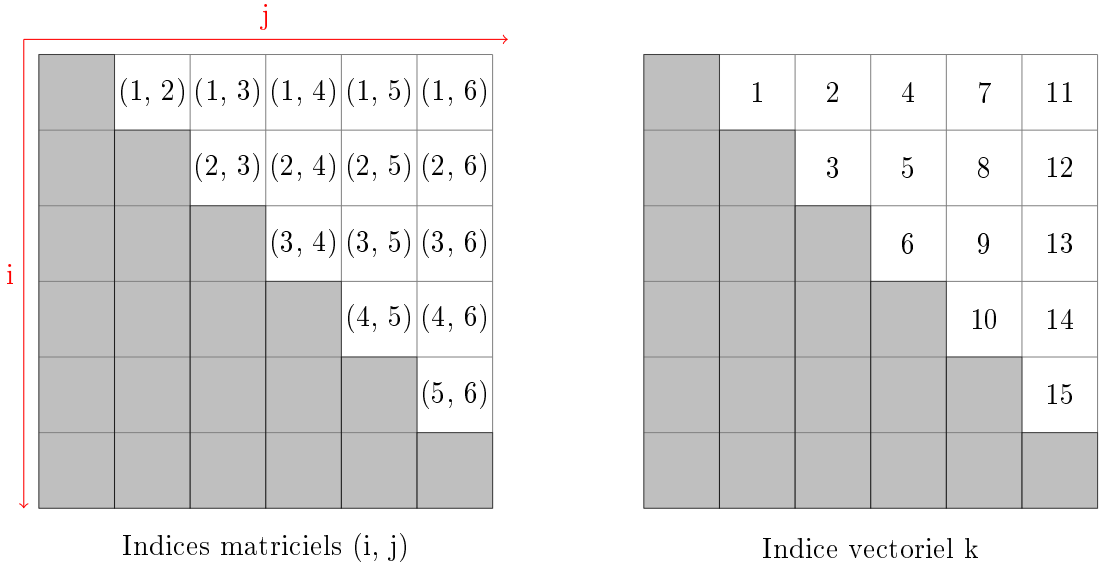


FIGURE 7.3 – Transformation d'une matrice triangulaire supérieure stricte en vecteur.

cas des matrices triangulaires inférieures et ϕ_4 et ψ_4 les transformations pour le cas des matrices triangulaires inférieures strictes.

7.2.1 Préliminaires : Nombres triangulaires

7.2.1.1 Définition géométrique

Un nombre triangulaire est un nombre entier non nul égal au nombre de pastilles rondes de même taille inscrite dans un triangle équilatéral construit comme indiqué en figure 7.5. Les sept premiers nombres triangulaires sont donc 1, 3, 6, 10, 15, 21, 28. On note par la suite t_n le nombre triangulaire de rang n. Le $n^{ème}$ nombre triangulaire t_n est le nombre de pastilles composant un triangle équilatéral de côté de longueur n pastilles. La figure montre la construction pour les premiers nombres triangulaires.

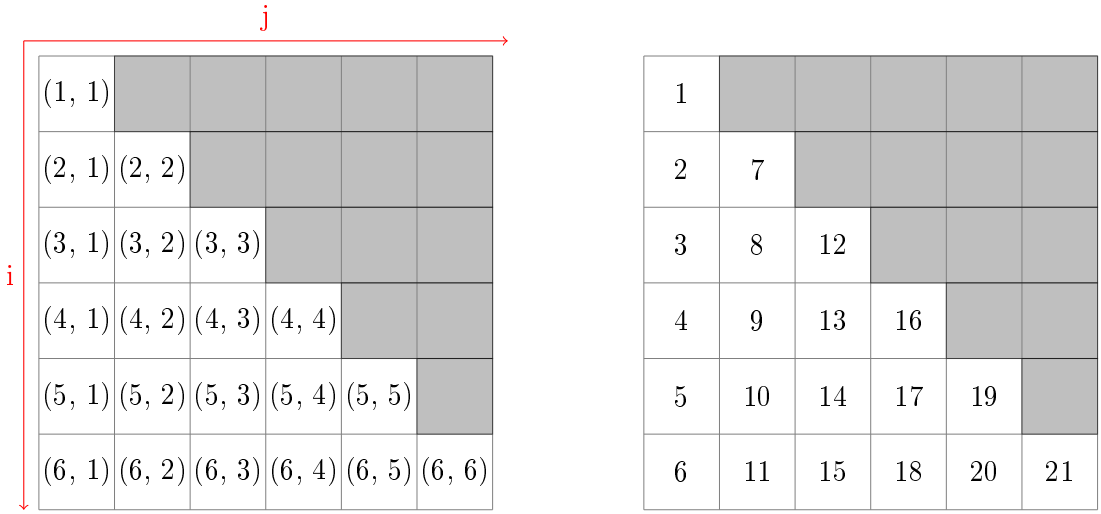


FIGURE 7.4 – Transformation d'une matrice triangulaire inférieure en vecteur.

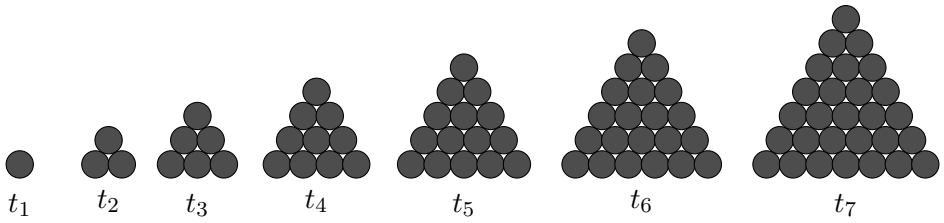


FIGURE 7.5 – Premiers nombres triangulaires.

7.2.1.2 Définition par récurrence

Plus formellement, les nombres triangulaires se définissent par une relation de récurrence donnée par l'équation 7.1. La figure 7.6 illustre la construction de t_n à partir des t_j, $j \in \{1, \ldots, n\}$.

$$\begin{cases} t_1 & = 1 \\ \forall n > 1, t_n & = t_{n-1} + n \end{cases} \tag{7.1}$$

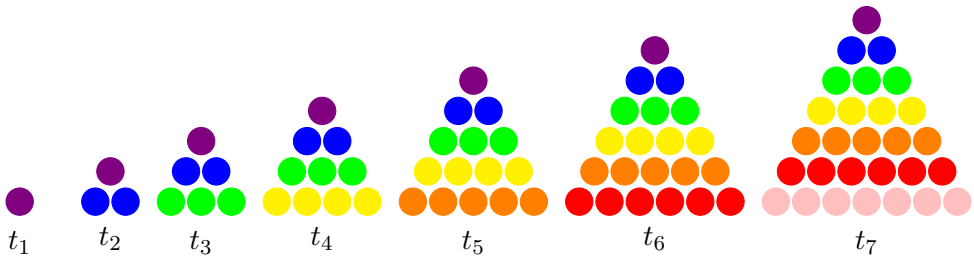


FIGURE 7.6 – Relation de récurrences entre les t_n.

La relation de récurrence peut être développée pour obtenir l'expression arithmétique de t_n.

$$\begin{aligned} t_n &= t_{n-1} + n \\ t_n &= t_{n-2} + n - 1 + n \\ &\vdots \\ t_n &= 1 + 2 + \ldots + n - 1 + n \\ t_n &= \sum_{i=1}^{n} i \end{aligned} \tag{7.2}$$

On peut ainsi trouver l'expression analytique de t_n donnée par :

$$\forall n \geq 1,\ t_n = \frac{n(n+1)}{2} \quad .$$

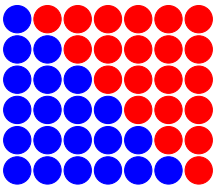


FIGURE 7.7 – Démonstration graphique.

Démonstration. La figure 7.7 démontre la formule de manière graphique. Sur la figure, ont été réarrangés deux triangles correspondant à t_n pour former un rectangle de taille $n \times (n+1)$. Est également donnée une preuve mathématique.

L'idée est d'écrire la formule donnant t_n une fois en écrivant la somme dans l'ordre croissant et une seconde fois en écrivant la somme dans l'ordre décroissant.

$$t_n = 1 + 2 + \ldots + n - 1 + n \tag{7.3}$$
$$t_n = n + n - 1 + \ldots + 2 + 1 \tag{7.4}$$

En additionnant 7.3 et 7.4, on obtient :

$$2t_n = (n+1) + (n+1) + \ldots + (n+1) \tag{7.5}$$
$$2t_n = n(n+1) \tag{7.6}$$

□

7.2.1.3 Quelques propriétés des nombres triangulaires

Propriété 1. *La somme de deux nombres triangulaires consécutifs est un carré parfait.*

Démonstration.

$$\begin{aligned} \forall n \geq 1,\ t_n + t_{n+1} &= \frac{n(n+1)}{2} + \frac{(n+1)(n+2)}{2} \\ &= (n+1)\left(\frac{n}{2} + \frac{n+2}{2}\right) \\ &= (n+1)^2 \end{aligned}$$

□

Propriété 2. *Si t_n est un nombre triangulaire alors $8t_n + 1$ est un carré parfait.*

Démonstration.

$$\begin{aligned}\forall n \geq 1,\ 8t_n + 1 &= 8\frac{n(n+1)}{2} \\ &= 4n^2 + n + 1 \\ &= (2n+1)^2\end{aligned}$$

□

7.2.1.4 Racine triangulaire

Par analogie avec la racine carrée, on peut définir une racine triangulaire.

Définition 3. *On définit la racine triangulaire d'un nombre x, notée par $\sqrt[T]{x}$, l'unique nombre tel que*

$$t_{\sqrt[T]{x}} = \frac{\sqrt[T]{x}(\sqrt[T]{x}+1)}{2} = x$$

Propriété 4.

$$\sqrt[T]{x} = \frac{\sqrt{8x+1}-1}{2} \ .$$

Démonstration. Calculons n en fonction du nombre triangulaire t_n.

$$\begin{aligned}2t_n &= n(n+1) \\ 2t_n &= n^2 + n \\ n^2 + n - 2t_n &= 0\end{aligned}$$

On obtient une équation du second degré de discriminant $\Delta = 1 + 8t_n$. Or la proposition 2 a établi que $8t_n+1$ est un carré parfait donc le discriminant est positif. Il existe alors deux solutions

$$n_1 = \frac{-1-\sqrt{8t_n+1}}{2} \quad n_2 = \frac{-1+\sqrt{8t_n+1}}{2} \ .$$

Mais $n_2 < 0$ n'est pas possible donc

$$n = \frac{-1+\sqrt{8t_n+1}}{2} \ .$$

□

7.2.2 Cas des matrices triangulaires supérieures

7.2.2.1 Transformation ϕ_1 des indices matriciels vers l'indice vectoriel

Établissons maintenant le lien entre les nombres triangulaires et l'indice vectoriel. Reprenons la transformation des indices matriciels vers l'indice vectoriel, rappelée à la figure 7.8. Sur la figure, est mise en évidence la diagonale principale. Les indices vectoriels des cases de la diagonale principale sont 1, 3, 6, 10, 15 et 21. Ce sont les premiers nombres triangulaires.

Superposons maintenant les triangles construisant les nombres triangulaires présentés en figure 7.6 sur la matrice de distances. La superposition est illustrée en figure 7.9. On voit apparaître les différents triangles. Lorsque la dimension de la matrice triangulaire passe de n colonnes et n lignes à $(n+1)$ colonnes et $(n+1)$ lignes, on ajoute $n+1$ cases au vecteur. La dernière case ainsi ajoutée a pour indice $k = t_{n+1}$. L'indice de la colonne j donne donc la taille du triangle auquel appartient l'indice vectoriel. En d'autres mots, on a que

$$\forall (i,j) \setminus i \geq 1,\ j \geq 1, \phi_1(i,j) \leq t_j \ .$$

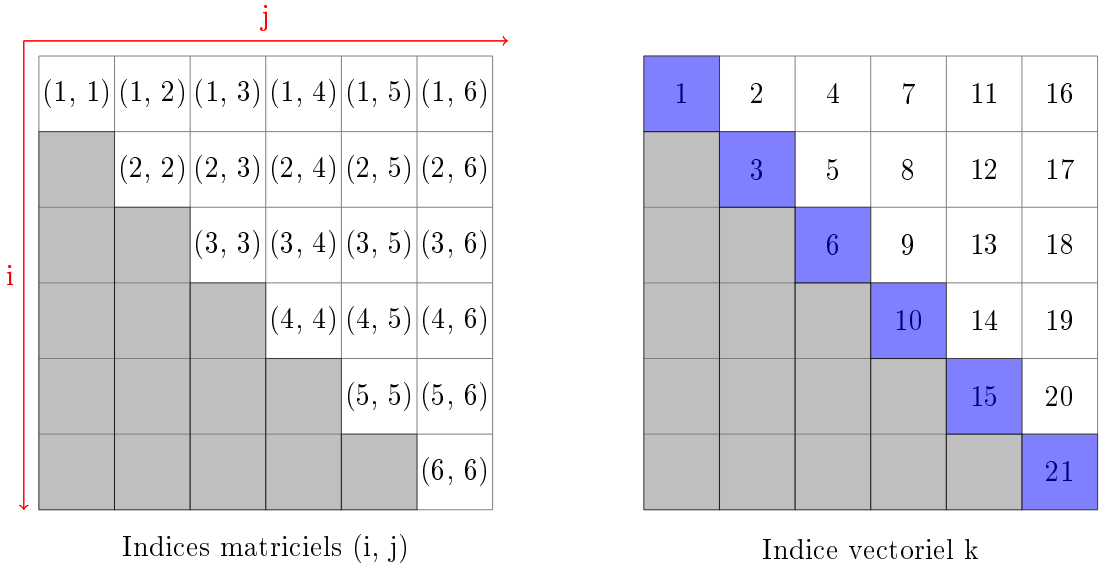


FIGURE 7.8 – Mise en évidence des nombres triangulaires parmi les indices vectoriels.

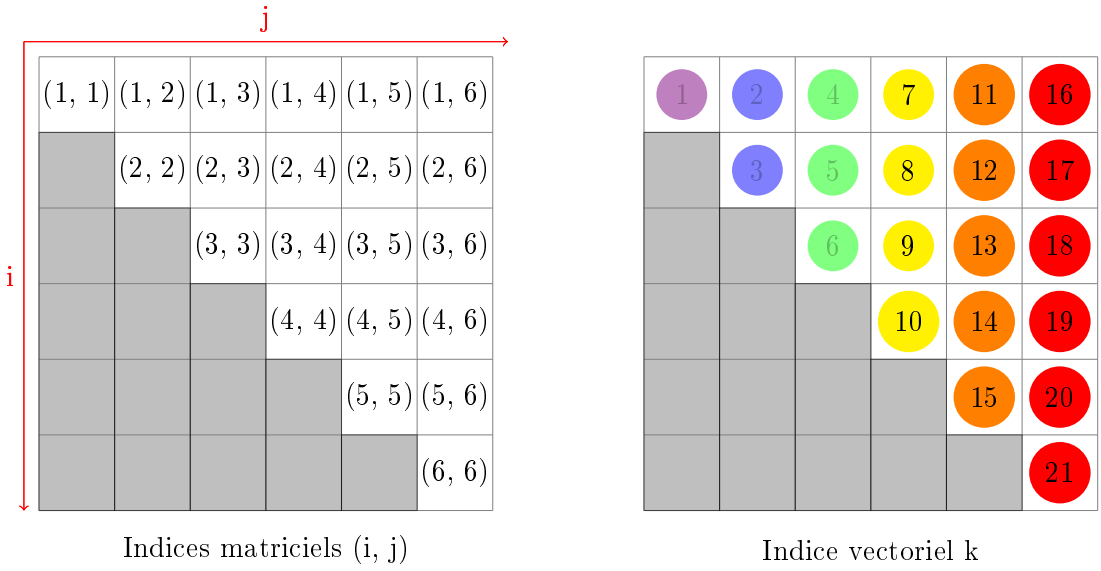


FIGURE 7.9 – Construction du vecteur de distances à l'aide des triangles de construction des t_n.

Puisque lors de la construction du vecteur à partir de la matrice, on ajoute les colonnes dans l'ordre croissant, la colonne j est rangée après les colonnes 1 à $j-1$. Les indices vectoriels de la colonne j sont donc supérieurs à t_{j-1}, qui est la valeur de la dernière case du triangle formé par les colonnes 1 à $j-1$. Autrement dit, on a :

$$\forall (i,j) \setminus i \geq 1,\, j \geq 1, t_{j-1} < \phi_1(i,j) \leq t_j \quad .$$

Lorsqu'on range la colonne j, on parcourt les lignes dans l'ordre croissant : on range d'abord $(1,j)$ puis $(2,j)$ jusqu'à ranger (j,j). Lorsqu'on ajoute la colonne j, la dernière case rangée dans le vecteur est la case $(j-1,j-1)$ avec un indice $\phi_1(i,j) = k = t_{j-1}$. La case (i,j), $i \leq j$ se range donc à l'indice $\phi_1(i,j) = k = t_{j-1} + i$.

Pour être complet, on peut traiter le cas où on considère une case dans la partie triangulaire inférieure. La case (i,j) de la partie triangulaire inférieure correspond par symétrie à la case (j,i) qui appartient bien à la matrice triangulaire supérieure. La case (i,j), $j \leq i$ se range donc à l'indice $\phi_1(i,j)k = t_{i-1} + j$.

En distinguant les deux cas, on peut écrire :

$$\phi_1 : \begin{cases} \mathbb{N}^2 \longrightarrow \mathbb{N} \\ (i,j) \longmapsto k = \begin{cases} i + \dfrac{j(j-1)}{2} \text{ si } i \leq j \\ j + \dfrac{i(i-1)}{2} \text{ si } j \leq i \end{cases} \end{cases} .$$

Finalement, la transformation des indices matriciels pour une matrice triangulaire vers des indices vectoriels est donnée par :

$$\phi_1 : \begin{cases} \mathbb{N}^2 \longrightarrow \mathbb{N} \\ (i,j) \longmapsto k = \min(i,j) + \frac{\max(i,j)(\max(i,j)-1)}{2} \end{cases} .$$

7.2.2.2 Transformation ψ_1 de l'indice vectoriel vers les indices matriciels

Établissons maintenant la réciproque de la fonction ϕ_1 pour retrouver les indices matriciels à partir de l'indice vectoriel. Par construction du vecteur, on a la propriété :

$$\forall (i,j) \setminus i \geq 1,\, j \geq 1, t_{j-1} < \phi_1(i,j) \leq t_j \ .$$

Utilisons maintenant la racine triangulaire sur cette équation. L'équation 7.7 est obtenue en rappelant que la racine triangulaire d'un nombre triangulaire est son rang $\sqrt[T]{t_n} = n$.

$$\begin{aligned} & t_{j-1} < k \leq t_j \\ \Longrightarrow & \sqrt[T]{t_{j-1}} < \sqrt[T]{k} \leq \sqrt[T]{t_j} \\ \Longleftrightarrow & j-1 < \sqrt[T]{k} \leq j \end{aligned} \tag{7.7}$$

On reconnaît à l'équation 7.7 la définition de la partie entière supérieure. On a donc

$$j = \lceil \sqrt[T]{k} \rceil \ .$$

On peut ensuite définir l'expression de i à partir de la définition de ϕ_1. Il suffit d'exprimer i en fonction de k et de j qu'on vient de déterminer.

$$\begin{aligned} \phi_1(i,j) = k &= i + \frac{j(j-1)}{2} \\ \Leftrightarrow i &= k - \frac{j(j-1)}{2} \end{aligned}$$

La transformation ψ_1 de l'indice vectoriel vers le couple d'indices matriciels est donnée par :

$$\psi_1 : \begin{cases} \mathbb{N} \longrightarrow \mathbb{N}^2 \\ k \longmapsto (i,j) = \left(k - \frac{1}{2}\left\lceil \frac{\sqrt{8k+1}-1}{2} \right\rceil \left(\left\lceil \frac{\sqrt{8k+1}-1}{2} \right\rceil - 1 \right), \left\lceil \frac{\sqrt{8k+1}-1}{2} \right\rceil \right) \end{cases} ,$$

Théorème 5. *Sur l'ensemble $(i,j) \in \mathbb{N}^2 \setminus i \leq j$, on a :*

$$\psi_1 = \phi_1^{-1}$$

Démonstration. Notons $Id_{\mathbb{N}}$ la fonction identité de $\mathbb{N}$ qui à tout entier n associe lui-même.

Notons également $Id_{\mathbb{N}^2}$ la fonction identité de $\mathbb{N}^2$ qui à tout couple d'entier (n_1, n_2) associe le couple (n_1, n_2).

Montrons d'abord que $\phi_1 \circ \psi_1 = Id_{\mathbb{N}}$. Prenons k$\in \mathbb{N}$, on a

$$\phi_1 \circ \psi_1(k) = \phi_1 \left(k - \frac{1}{2}\left\lceil \frac{\sqrt{8k+1}-1}{2} \right\rceil \left(\left\lceil \frac{\sqrt{8k+1}-1}{2} \right\rceil - 1 \right), \left\lceil \frac{\sqrt{8k+1}-1}{2} \right\rceil \right)$$

$$\phi_1 \circ \psi_1(k) = k - \frac{1}{2}\left\lceil \frac{\sqrt{8k+1}-1}{2} \right\rceil \left(\left\lceil \frac{\sqrt{8k+1}-1}{2} \right\rceil - 1 \right) + \frac{1}{2}\left\lceil \frac{\sqrt{8k+1}-1}{2} \right\rceil \left(\left\lceil \frac{\sqrt{8k+1}-1}{2} \right\rceil - 1 \right)$$

$$\phi_1 \circ \psi_1(k) = k$$

Montrons maintenant que $\psi_1 \circ \phi_1 = Id_{\mathbb{N}^2}$. Prenons $(i,j) \in \mathbb{N}^2$ avec $i \leq j$, on a alors :

$$\psi_1 \circ \phi_1(i,j) = \psi\left(i + \frac{j(j-1)}{2}\right)$$

$$\psi_1 \circ \phi_1(i,j) = \left(i + \frac{j(j-1)}{2} - \frac{1}{2}\left\lceil \frac{\sqrt{8\left(i+\frac{j(j-1)}{2}\right)+1}-1}{2} \right\rceil \left(\left\lceil \frac{\sqrt{8\left(i+\frac{j(j-1)}{2}\right)+1}-1}{2} \right\rceil - 1 \right) , \left\lceil \frac{\sqrt{8\left(i+\frac{j(j-1)}{2}\right)+1}-1}{2} \right\rceil \right)$$

On reconnaît la racine triangulaire introduite en section 7.2.1.4 :

$$\left\lceil \frac{\sqrt{8\left(i+\frac{j(j-1)}{2}\right)+1}-1}{2} \right\rceil = \sqrt[T]{i + \frac{j(j-1)}{2}} \ .$$

Calculons cette racine triangulaire. Observons d'abord que

$$i + \frac{j(j-1)}{2} = i + t_{j-1} \ .$$

On peut donc encadrer ce nombre par deux nombres triangulaires.

$$\begin{aligned} & t_{j-1} < i + t_{j-1} \leq t_j \\ \iff & \sqrt[T]{t_{j-1}} < \sqrt[T]{i + t_{j-1}} \leq \sqrt[T]{t_j} \\ \iff & j - 1 < \sqrt[T]{i + t_{j-1}} \leq j \end{aligned}$$

À la dernière équation, on reconnaît une des propriétés de la partie entière supérieure. On a donc

$$j = \left\lceil \sqrt[T]{i + t_{j-1}} \right\rceil \ .$$

Reprenons maintenant le calcul de $\psi_1 \circ \phi_1(i,j)$

$$\psi_1 \circ \phi_1(i,j) = \left(i + \frac{j(j-1)}{2} - \frac{1}{2}\left\lceil \frac{\sqrt{8\left(i+\frac{j(j-1)}{2}\right)+1}-1}{2} \right\rceil \left(\left\lceil \frac{\sqrt{8\left(i+\frac{j(j-1)}{2}\right)+1}-1}{2} \right\rceil - 1 \right) , \left\lceil \frac{\sqrt{8\left(i+\frac{j(j-1)}{2}\right)+1}-1}{2} \right\rceil \right)$$

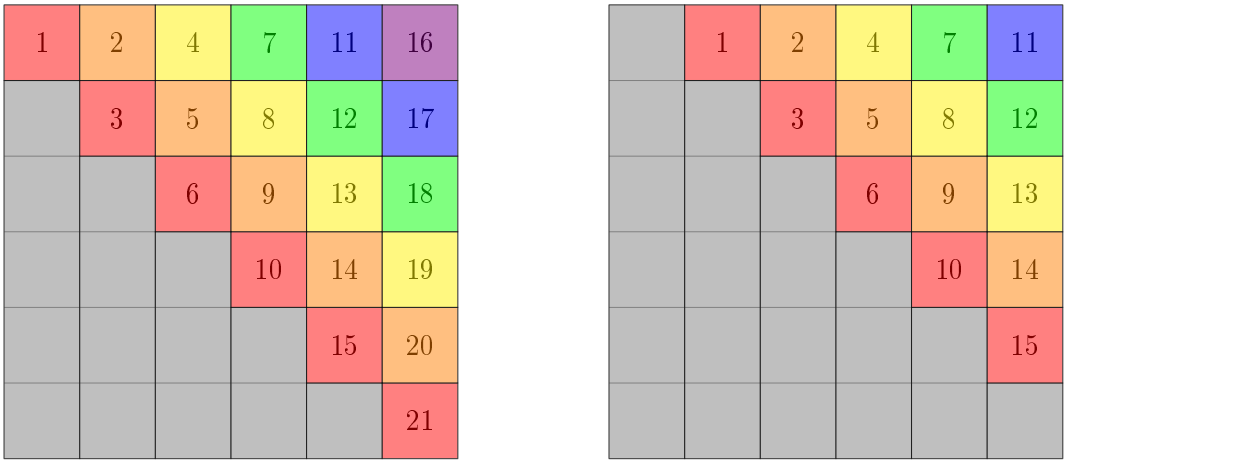
$$\psi_1 \circ \phi_1(i,j) = \left(i + \frac{j(j-1)}{2} - \frac{1}{2}j(j-1), j \right)$$

$$\psi_1 \circ \phi_1(i,j) = (i,j)$$

□

7.2.3 Cas des matrices triangulaires supérieures strictes

Considérons maintenant le cas d'une matrice triangulaire stricte. Observons que par rapport au cas de la matrice triangulaire, les indices vectoriels sont décalés d'une diagonale supérieure, comme le montre la figure 7.10.



Indice vectoriel k pour une matrice triangulaire — Indice vectoriel k pour une matrice triangulaire stricte

FIGURE 7.10 – Décalage pour le cas d'une matrice triangulaire stricte par rapport au cas d'une matrice triangulaire.

Si la matrice triangulaire stricte supérieure a n colonnes et lignes, le vecteur aura t_{n-1} cases. La colonne j de la matrice triangulaire stricte se range à la place de la colonne $j-1$ pour une matrice triangulaire supérieure.

La colonne j d'une matrice triangulaire supérieure stricte est rangée avec les mêmes numéros que la colonne $j-1$ de la matrice triangulaire. Soit un couple d'indice matriciel (i,j), on a que $k = \phi_2(i,j) = \phi_1(i,j-1)$ et donc en composant par ψ_1, on obtient $(i,j-1) = \psi_1(k)$. Les transformations sont donc :

$$\phi_2 : \begin{cases} \mathbb{N}^2 \longrightarrow \mathbb{N} \\ (i,j) \longmapsto k = \min(i,j) + \frac{(\max(i,j)-1)(\max(i,j)-2)}{2} \end{cases} .$$

$$\psi_2 : \begin{cases} \mathbb{N} \longrightarrow \mathbb{N}^2 \\ k \longmapsto (i,j) = \left(k - \frac{1}{2}\left(\left\lceil \frac{\sqrt{8k+1}-1}{2} \right\rceil\right)\left(\left\lceil \frac{\sqrt{8k+1}-1}{2} \right\rceil - 1\right), \left\lceil \frac{\sqrt{8k+1}-1}{2} \right\rceil + 1\right) \end{cases} ,$$

7.2.4 Cas d'une matrice triangulaire inférieure

La figure 7.11 rappelle le rangement d'une matrice triangulaire inférieure dans un vecteur. Contrairement au cas d'une matrice triangulaire supérieure, les nombres triangulaires ne sont pas placés dans un endroit particulier (cases en bleu). L'idée est de transformer l'indice vectoriel k pour mieux placer les nombres triangulaires. Pour cela, on propose de prendre $k' = \frac{n(n+1)}{2} - k$. En appliquant cette transformation f, on obtient les indices vectoriels montrés en figure 7.12 : les nombres triangulaires (dans les cases bleues) sont ainsi placés sur la dernière ligne. On remarque que la dernière case a pour indice 0 qui peut se définir comme le nombre triangulaire t_0.

7.2.4.1 Transformation ϕ_3 des indices matriciels vers l'indice vectoriel

Maintenant que les nombres triangulaires sont bien placés, on peut superposer les indices aux triangles de construction des nombres triangulaires. Cette superposition est montrée en figure 7.13.

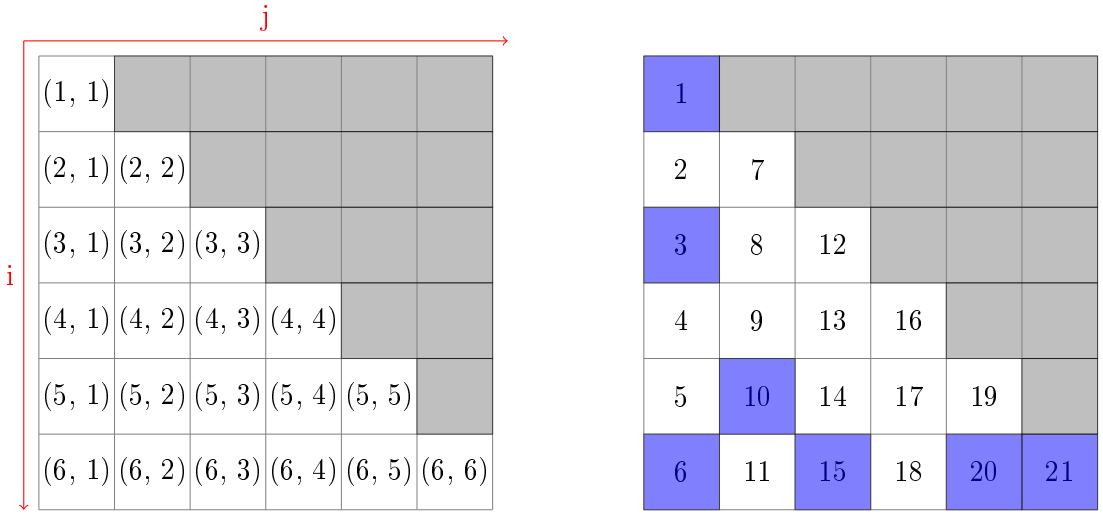


FIGURE 7.11 – Transformation d'une matrice triangulaire inférieure en vecteur.

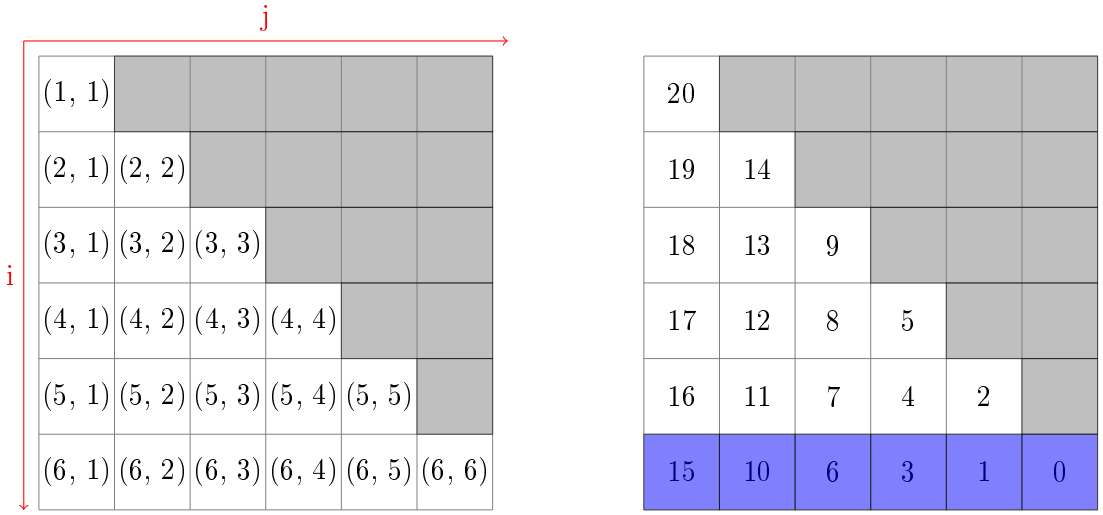


FIGURE 7.12 – Transformation de l'indice vectoriel : $k' = \frac{n(n+1)}{2} - k$.

On remarque que la colonne j correspond au triangle de construction de t_{n-j+1} avec un décalage des indices de 1. La colonne j est ainsi numérotée de t_{n-j} à $t_{n-j+1} - 1$, avec pour la dernière colonne $k' = 0$. Une fois dans la colonne j, les lignes sont rangées dans l'ordre décroissant : la ligne n est rangée en premier et la ligne 1 est rangé en dernier. Lorsqu'on range la ligne i, on ajoute à $t_{n-(j-1)}$ la valeur $n - i$. On a donc que $k' = t_{n-j} + n - i$.

Connaissant la transformation f de k vers k', on peut obtenir l'expression de $k = \phi_3(i, j)$ en fonction de (i, j) puisque $k = \frac{n(n+1)}{2} - k'$. La transformation ϕ_3 pour une matrice triangulaire inférieure est donnée par :

$$\phi_3 : \begin{cases} \mathbb{N}^2 \longrightarrow \mathbb{N} \\ (i, j) \longmapsto k = \frac{n(n-1)}{2} - \frac{(n-j)(n+1-j)}{2} + i \end{cases} .$$

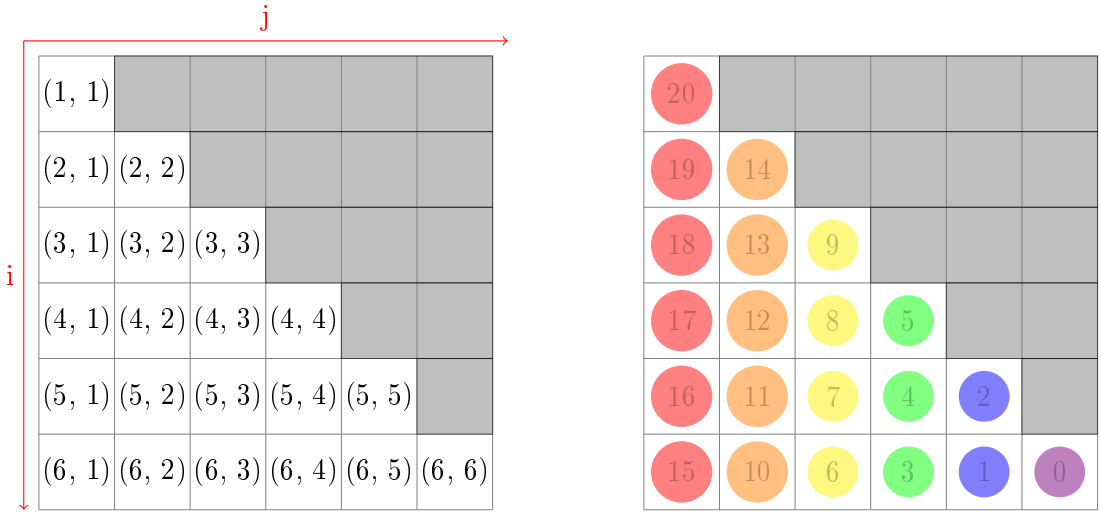


FIGURE 7.13 – Mise en relation entre l'indice k' et les triangles de construction des t_n.

7.2.4.2 Transformation ψ_3 de l'indice vectoriel vers les indices matriciels

Établissons maintenant la fonction ψ_3 réciproque de ϕ_3 permettant de passer de l'indice vectoriel aux indices matriciels.

Rappelons que pour déterminer ϕ_3, on applique d'abord la transformation f à k pour obtenir $f(k) = k' = \frac{n(n+1)}{2} - k$. On a montré à la sous-section précédente que par construction, on a :

$$t_{n-j} \leq k' < t_{n-j+1} \ .$$

Prenons la racine triangulaire de cette équation. On rappelle que $\sqrt[T]{t_n} = n$. On obtient alors

$$n - j \leq \sqrt[T]{k'} < n - j + 1 \ .$$

On reconnaît dans l'équation ci-dessus la définition de la partie entière. On a ainsi :

$$n - j = \lfloor \sqrt[T]{k'} \rfloor \Leftrightarrow j = n - \lfloor \sqrt[T]{k'} \rfloor \ .$$

Maintenant qu'on a déterminé j, on peut calculer i à partir de l'expression de ϕ_3.

$$\begin{aligned} & k' = t_{n-j} + n - i \\ \Leftrightarrow\ & i = t_{n-j} + n - k' \end{aligned}$$

La transformation ψ_3 de l'indice vectoriel vers le couple d'indices matriciels est donnée par :

$$\psi_3 : \begin{cases} \mathbb{N} \longrightarrow \mathbb{N}^2 \\ k \longmapsto (i,j) = \left(n - k' + \frac{1}{2} \left\lfloor \frac{\sqrt{8k'+1}-1}{2} \right\rfloor \left(\left\lfloor \frac{\sqrt{8k'+1}-1}{2} \right\rfloor + 1 \right), n - \left\lfloor \frac{\sqrt{8k'+1}-1}{2} \right\rfloor \right) \end{cases} ,$$

ce qui donne en remplaçant l'expression de k' :

$$\psi_3 : \begin{cases} \mathbb{N} \longrightarrow \mathbb{N}^2 \\ k \longmapsto (i,j) = \left(k + \frac{n(1-n)}{2} + \frac{1}{2} \left\lfloor \frac{\sqrt{8\left(\frac{n(n+1)}{2} - k\right)+1}-1}{2} \right\rfloor \left(\left\lfloor \frac{\sqrt{8\left(\frac{n(n+1)}{2} - k\right)+1}-1}{2} \right\rfloor + 1 \right), \right. \\ \qquad \left. n - \left\lfloor \frac{\sqrt{8\left(\frac{n(n+1)}{2} - k\right)+1}-1}{2} \right\rfloor \right) \end{cases} .$$

Théorème 6. *Sur l'ensemble* $(i,j) \in \mathbb{N}^2 \backslash j \leq i$, *on a :*

$$\psi_3 = \phi_3^{-1}$$

Démonstration. Notons $Id_{\mathbb{N}}$ la fonction identité de $\mathbb{N}$ qui à tout entier n associe lui-même.

Notons également $Id_{\mathbb{N}^2}$ la fonction identité de $\mathbb{N}^2$ qui à tout couple d'entier (n_1, n_2) associe le couple (n_1, n_2).

Montrons d'abord que $\phi_3 \circ \psi_3 = Id_{\mathbb{N}}$. Prenons k$\in \mathbb{N}$, en notant $k' = \frac{n(n+1)}{2} - k$ et en rappelant que $\sqrt[T]{k'} = \frac{\sqrt{8k'+1}-1}{2}$ on a

$$\begin{aligned}
\phi_3 \circ \psi_3(k) &= \phi_3\left(n - k' + \frac{1}{2}\sqrt[T]{k'}\left(\sqrt[T]{k'}+1\right), n - \sqrt[T]{k'}\right) \\
\phi_3 \circ \psi_3(k) &= \frac{n(n-1)}{2} - \frac{1}{2}(n-(n-\sqrt[T]{k'}))(n+1-(n-\sqrt[T]{k'}r)) + n - k' + \frac{\sqrt[T]{k'}(\sqrt[T]{k'}+1)}{2}) \\
\phi_3 \circ \psi_3(k) &= \frac{n(n+1)}{2} - \frac{\sqrt[T]{k'}(\sqrt[T]{k'}+1)}{2} - k' + \frac{\sqrt[T]{k'}(\sqrt[T]{k'}+1)}{2} \\
\phi_3 \circ \psi_3(k) &= \frac{n(n+1)}{2} - k' = k
\end{aligned}$$

Montrons maintenant que $\psi_1 \circ \phi_1 = Id_{\{}1, \ldots, n\}^2$.

Prenons $(i,j) \in \{1, \ldots, n\}^2$ avec $j \leq i$, on a alors :

$$\begin{aligned}
\psi_3 \circ \phi_3(i,j) &= \psi_3\left(\frac{n(n-1)}{2} - \frac{(n-j)(n+1-j)}{2} + i\right) \\
\psi_3 \circ \phi_3(i,j) &= \psi_3\left(\frac{n(n-1)}{2} - t_{n-j} + i\right) \\
\psi_3 \circ \phi_3(i,j) &= \left(\frac{n(1-n)}{2} + \frac{n(n-1)}{2} - t_{n-j} + i + \frac{1}{2}\sqrt[T]{n+t_{n-j}-i}\left(\sqrt[T]{n+t_{n-j}-i}+1\right),\right. \\
&\qquad \left. n - \sqrt[T]{n+t_{n-j}-i}\right) \\
\psi_3 \circ \phi_3(i,j) &= \left(i - t_{n-j} + \frac{1}{2}\sqrt[T]{n+t_{n-j}-i}\left(\sqrt[T]{n+t_{n-j}-i}+1\right), n - \sqrt[T]{n+t_{n-j}-i}\right)
\end{aligned}$$

Calculons maintenant la racine triangulaire de $\sqrt[T]{n-i+t_{n-j}}$. Comme $i \leq n$, on a $n-i \geq 0$. Donc

$$t_{n-j} \leq n - i + t_{n-j} \ .$$

Montrons ensuite que $n - i + t_{n-j} < t_{n+1-j}$.

$$\begin{aligned}
t_{n+1-j} - (t_{n-j} + n - i) &= \frac{(n+1-j)(n+2-j)}{2} - \frac{n(n+1)}{2} - n + i \\
&= i + 1 - j
\end{aligned}$$

Or on a pris $j \leq i$ avec $(i,j) \in \mathbb{N}^2$ donc $i > j - 1$. On a finalement

$$t_{n-j} \leq n - i + t_{n-j} < t_{n+1-j}$$

On reconnaît une des propriétés de la partie entière. On a donc

$$n - j = \left\lfloor \sqrt[T]{n-i+t_{n-j}} \right\rfloor \ .$$

Reprenons maintenant le calcul de $\psi_3 \circ \phi_3$:

$$\psi_3 \circ \phi_3(i,j) = \left(i - t_{n-j} + i + \frac{1}{2}(n-j)(n+1-j), j \right)$$
$$\psi_3 \circ \phi_3(i,j) = (i,j)$$

□

7.2.5 Cas d'une matrice triangulaire inférieure stricte

Considérons maintenant le cas d'une matrice triangulaire inférieure stricte. Par rapport au cas développé en section 7.2.4, la diagonale n'est plus stockée. La figure 7.14 montre la transformation d'une matrice triangulaire inférieure stricte en vecteur.

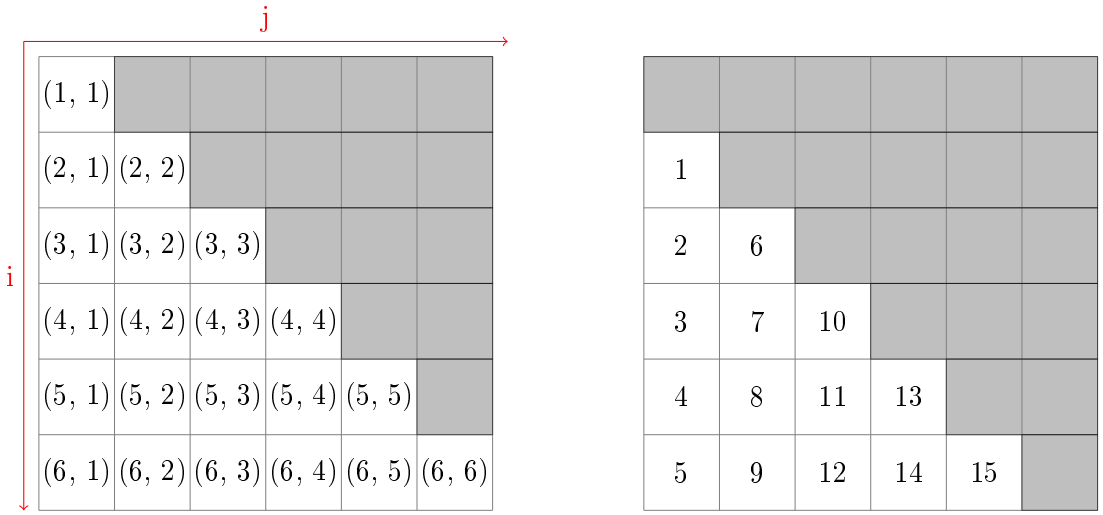


FIGURE 7.14 – Transformation d'une matrice triangulaire inférieure stricte en vecteur.

Malheureusement, contrairement aux cas des matrices triangulaires supérieures, il n'apparaît ici de lien évident entre l'indice k pour une matrice triangulaire inférieure stricte et l'indice k pour une matrice triangulaire inférieure (montrée en Figure 7.11). Pourtant, on peut se ramener au cas d'une matrice triangulaire inférieure. Si la matrice triangulaire inférieure stricte est de taille n ($n = 6$ sur la figure 7.14), on considère une matrice triangulaire inférieure (avec la diagonale) de taille $n-1$. La figure 7.15 montre une telle matrice pour $n = 5$. Appelons (i^*, j^*) les indices matriciels de la matrice inférieure de taille $n^* = n-1$ et k^* son indice vectoriel. On remarque que l'indice k^* correspond à l'indice k qu'on souhaite avoir dans le cas d'une matrice triangulaire inférieure stricte. La formule donnant k^* donne donc l'indice k cherché. Reste maintenant à trouver la relation entre (i^*, j^*) et (i, j). Si on compare les figures 7.14 et 7.15, on constate qu'on a les mêmes numéros de colonnes donc $j^* = j$. En revanche, la ligne $i = 2$ correspond à la ligne $i^* = 1$: on a donc un décalage des ligne $i^* = i - 1$. En remplaçant i, j^* et n^* par leur valeur en fonction de i, j et n dans l'expression de ϕ_3 et ψ_3, on obtient les transformations suivantes :

$$\phi_4 : \begin{cases} \mathbb{N}^2 \longrightarrow \mathbb{N} \\ (i,j) \longmapsto k = \frac{(n-1)(n-2)}{2} - \frac{(n-1-j)(n-j)}{2} + i - 1 \end{cases} .$$

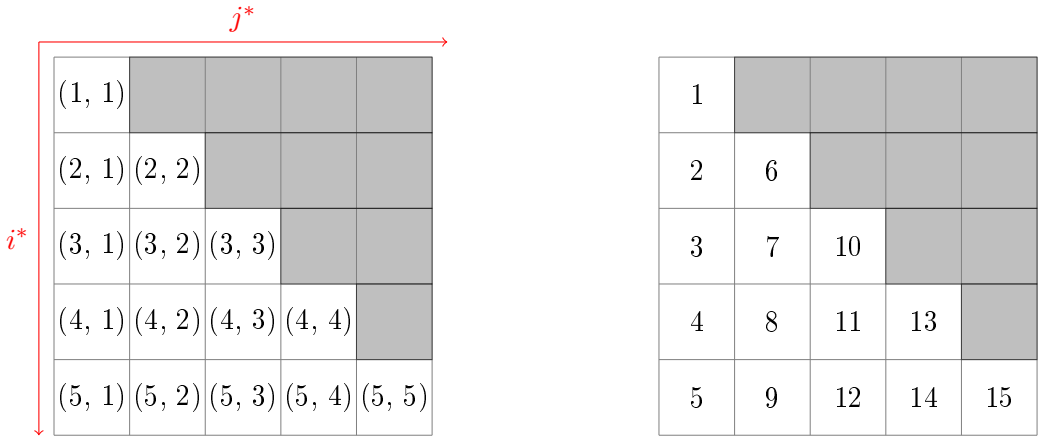


FIGURE 7.15 – Transformation d'une matrice triangulaire inférieure de taille $(n-1)$ en vecteur.

$$\psi_4 : \begin{cases} \mathbb{N} \longrightarrow \mathbb{N}^2 \\ k \longmapsto (i,j) \end{cases} = \left(k + \frac{(n-1)(2-n)}{2} + \frac{1}{2} \left\lfloor \frac{\sqrt{8\left(\frac{n(n-1)}{2} - k\right)+1}-1}{2} \right\rfloor \left(\left\lfloor \frac{\sqrt{8\left(\frac{n(n-1)}{2} - k\right)+1}-1}{2} \right\rfloor + 1 \right), \right.$$
$$\left. n - \left\lfloor \frac{\sqrt{8\left(\frac{n(n-1)}{2} - k\right)+1}-1}{2} \right\rfloor \right) .$$

L'obtention de i dans ψ_4 peut étonner au premier regard car il y a toujours n alors qu'on s'attend à voir $n-1$. Toutefois, il ne faut pas oublier que certes on a $n* = n-1$, mais également $i^* = i-1$ et d'où $i = i^* + 1 = n - 1 + 1 + \ldots$.

7.2.6 Stockage choisi

La figure 7.16 montre le stockage de la matrice de distances sous forme de vecteur : les cases grisées sont des cases ignorées lors du passage sous forme de vecteur et les numéros indiquent les indices dans le vecteur ***dist***. Nous avons choisi de stocker la matrice triangulaire supérieure stricte colonne par colonne. La section 7.2.2 établit la formule de l'indice k dans le vecteur ***dist*** en fonction des indices i et j de la matrice :

$$\forall i \neq j, k = \min(i,j) + \frac{(\max(i,j)-1)(\max(i,j)-2)}{2} .$$

Ainsi la colonne D_j est stockée aux indices

$$\{k = j + \frac{(i-1)(i-2)}{2}, i = 1, \ldots, j-1\} \bigcup \{k = i + \frac{(j-1)(j-2)}{2}, i = j+1, \ldots, n_V\} .$$

7.3 1ère version de la parallélisation

La version expliquée ci-dessous est la version présentée à la conférence PPAM2017 [10].

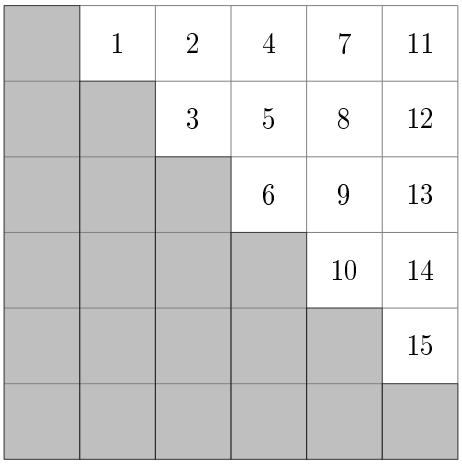


FIGURE 7.16 – Stockage choisi pour le vecteur de distance.

7.3.1 Parallélisation avec OpenMP

Pour le calcul de la distance, les tuiles sont distribuées entre les différents processus légers. Notons n_{CB} le nombre de tuiles. Prenons le processus léger k, il a la charge de la tuile $\boldsymbol{X}_{.k}$. Il doit calculer la distance entre les colonnes de la tuile $\boldsymbol{X}_{.k}$. Ensuite, le processus léger k calcule la distance entre les colonnes de la tuile $\boldsymbol{X}_{.k}$ et les colonnes des tuiles $\boldsymbol{X}_{.j}$ $j \in \{k+1, \ldots, n_{CB}\}$. On ne considère que les tuiles $\boldsymbol{X}_{.j}$ $j \in \{k, \ldots, n_{CB}\}$ par symétrie de la distance. La distance entre la tuile $\boldsymbol{X}_{.k}$ et les tuiles $\boldsymbol{X}_{.j}$ $j \in \{1, \ldots, k-1\}$ sont calculées par les processus légers en charge des tuiles $\boldsymbol{X}_{.j}$. Par exemple, la distance entre la tuile $\boldsymbol{X}_{.k}$ et la tuile $\boldsymbol{X}_{.k-1}$ est calculée par le processus léger en charge de $\boldsymbol{X}_{.k-1}$. La figure 7.17 illustre la répartition du calcul de la distance en OpenMP pour sept tuiles et quatre processus légers.

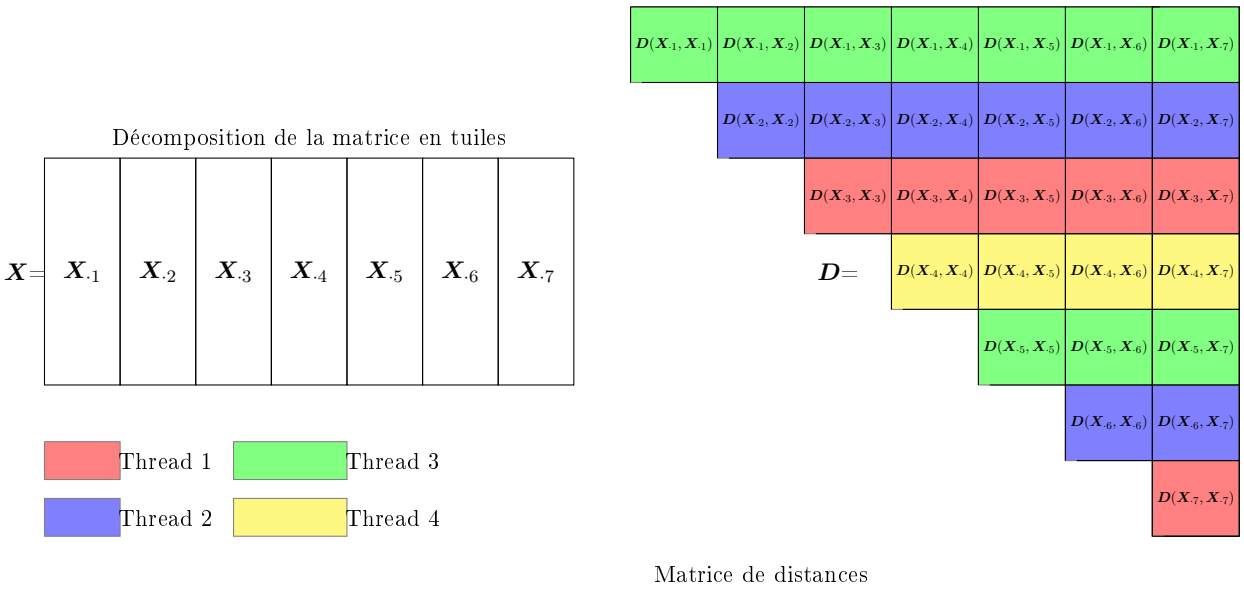


FIGURE 7.17 – $1^{ère}$ version : Exemple de calcul de la distance avec OpenMP pour 7 tuiles et 4 processus .

7.3.2 Parallélisation sur GPU avec CUDA

Comme on ne peut pas stocker tout le vecteur de distances dans la mémoire GPU, la stratégie de découpage en tuiles rectangulaires a aussi été adoptée pour le calcul sur GPU.

7.3.2.1 Formule matricielle de calcul de distance

Considérons les tuiles $\boldsymbol{A}$ de taille m_{CA} regroupant les colonnes d'indice i_1 à $i_{m_{CA}}$ et $\boldsymbol{B}$ de taille m_{CB} regroupant les colonnes d'indice j_1 à $j_{m_{CB}}$:

$$\boldsymbol{A} = \begin{pmatrix} \boldsymbol{x}_{i_1} & \dots & \boldsymbol{x}_{i_{m_{CA}}} \end{pmatrix} \quad \boldsymbol{B} = \begin{pmatrix} \boldsymbol{x}_{j_1} & \dots & \boldsymbol{x}_{j_{m_{CB}}} \end{pmatrix} \ .$$

Les tuiles sont en général de la même taille ($m_{CA} = m_{CB}$) sauf si le nombre de colonnes n_{V} n'est pas multiple de la taille de tuiles choisie. Définissons $\boldsymbol{s}$ le vecteur ligne contenant la norme euclidienne au carré des colonnes de $\boldsymbol{X}$:

$$\boldsymbol{s} = \left(\|\boldsymbol{x}_1\|_2^2, \dots, \|\boldsymbol{x}_{n_{\mathrm{V}}}\|_2^2 \right) \ .$$

Notons ensuite $\boldsymbol{s}_A$ la restriction du vecteur $\boldsymbol{s}$ aux colonnes de $\boldsymbol{A}$ et $\boldsymbol{s}_B$ la restriction du vecteur $\boldsymbol{s}$ aux colonnes de $\boldsymbol{B}$. La distance entre les colonnes de la tuile $\boldsymbol{A}$ est donnée par le théorème 7.3.1. La distance entre les colonnes de la tuile $\boldsymbol{A}$ et les colonnes de la tuile $\boldsymbol{B}$ est donnée par le théorème 7.3.2.

Théorème 7.3.1. $\boldsymbol{D}_A = \mathbf{1}_{\boldsymbol{m_{CA}}} \boldsymbol{s}_A + \boldsymbol{s}_A^\top \mathbf{1}_{\boldsymbol{m_{CA}}}^\top - 2\boldsymbol{A}^\top \boldsymbol{A}$.

Démonstration du théorème 7.3.1.

$$\boldsymbol{D}_A = \begin{pmatrix} 0 & \sum_{t=1}^{n_{\mathrm{T}}} (x_{ti_1} - x_{ti_2})^2 & \dots & \sum_{t=1}^{n_{\mathrm{T}}} \left(x_{ti_1} - x_{ti_{m_{CA}}} \right)^2 \\ \sum_{t=1}^{n_{\mathrm{T}}} (x_{ti_2} - x_{ti_1})^2 & 0 & \dots & \sum_{t=1}^{n_{\mathrm{T}}} \left(x_{ti_2} - x_{ti_{m_{CA}}} \right)^2 \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{t=1}^{n_{\mathrm{T}}} \left(x_{ti_{m_{CA}}} - x_{ti_1} \right)^2 & \sum_{t=1}^{n_{\mathrm{T}}} \left(x_{ti_{m_{CA}}} - x_{ti_2} \right)^2 & \dots & 0 \end{pmatrix}$$

$$\boldsymbol{D}_A = \begin{pmatrix} 0 & \dots & \sum_{t=1}^{n_{\mathrm{T}}} \left(x_{ti_1}^2 + x_{ti_{m_{CA}}}^2 - 2x_{ti_1} x_{ti_{m_{CA}}} \right) \\ \vdots & \ddots & \vdots \\ \sum_{t=1}^{n_{\mathrm{T}}} \left(x_{ti_{m_{CA}}}^2 + x_{ti_1}^2 - 2x_{ti_{m_{CA}}} x_{ti_1} \right) & \dots & 0 \end{pmatrix}$$

$$\boldsymbol{D}_A = \begin{pmatrix} \sum_{t=1}^{n_{\mathrm{T}}} x_{ti_1}^2 & \dots & \sum_{t=1}^{n_{\mathrm{T}}} x_{ti_1}^2 \\ \vdots & \ddots & \vdots \\ \sum_{t=1}^{n_{\mathrm{T}}} x_{ti_{m_{CA}}}^2 & \dots & \sum_{t=1}^{n_{\mathrm{T}}} x_{ti_{m_{CA}}}^2 \end{pmatrix} + \begin{pmatrix} \sum_{t=1}^{n_{\mathrm{T}}} x_{ti_1}^2 & \dots & \sum_{t=1}^{n_{\mathrm{T}}} x_{ti_{m_{CA}}}^2 \\ \vdots & \ddots & \vdots \\ \sum_{t=1}^{n_{\mathrm{T}}} x_{ti_1}^2 & \dots & \sum_{t=1}^{n_{\mathrm{T}}} x_{ti_{m_{CA}}}^2 \end{pmatrix}$$
$$- 2 \begin{pmatrix} \sum_{t=1}^{n_{\mathrm{T}}} x_{ti_1}^2 & \dots & \sum_{t=1}^{n_{\mathrm{T}}} x_{ti_1} x_{ti_{m_{CA}}} \\ \vdots & \ddots & \vdots \\ \sum_{t=1}^{n_{\mathrm{T}}} x_{ti_{m_{CA}}} x_{ti_1} & \dots & \sum_{t=1}^{n_{\mathrm{T}}} x_{ti_{m_{CA}}}^2 \end{pmatrix}$$

$$\boldsymbol{D}_A = \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} \times \begin{pmatrix} \sum_{t=1}^{n_{\mathrm{T}}} x_{ti_1}^2 & \dots & \sum_{t=1}^{n_{\mathrm{T}}} x_{ti_{m_{CA}}}^2 \end{pmatrix} + \begin{pmatrix} \sum_{t=1}^{n_{\mathrm{T}}} x_{ti_1}^2 \\ \vdots \\ \sum_{t=1}^{n_{\mathrm{T}}} x_{ti_{m_{CA}}}^2 \end{pmatrix} \times \begin{pmatrix} 1 & \dots & 1 \end{pmatrix}$$

$$- 2 \begin{pmatrix} x_{1i_1} & \dots & x_{n_{\mathrm{T}}i_1} \\ \vdots & \ddots & \vdots \\ x_{1i_m} & \dots & x_{n_{\mathrm{T}}i_{m_{CA}}} \end{pmatrix} \times \begin{pmatrix} x_{1i_1} & \dots & x_{1i_{m_{CA}}} \\ \vdots & \ddots & \vdots \\ x_{n_{\mathrm{T}}i_1} & \dots & x_{n_{\mathrm{T}}i_{m_{CA}}} \end{pmatrix}$$

$$\boldsymbol{D}_A = \mathbf{1}_{\boldsymbol{m_{CA}}} \boldsymbol{s}_A + \boldsymbol{s}_A^\top \mathbf{1}_{\boldsymbol{m_{CA}}}^\top - 2\boldsymbol{A}^\top \boldsymbol{A}$$

□

L'idée est ensuite d'utiliser le GPU pour calculer les produits matriciels à l'aide de CUBLAS. Les formules données dans les théorèmes 7.3.1 et 7.3.2 sont calculables avec trois appels à la fonction GEMM, qui calcule $C^{op_C} = \alpha A^{op_A} B^{op_B} + \beta C^{op_C}$ avec op_A, op_B et op_C indiquant si les matrices doivent être transposées ou non.

Théorème 7.3.2. $\boldsymbol{D}_{AB} = \mathbf{1}_{\boldsymbol{m_{CB}}} \boldsymbol{s}_B + \boldsymbol{s}_A^\top \mathbf{1}_{\boldsymbol{m_{CA}}}^\top - 2\boldsymbol{A}^\top \boldsymbol{B}$.

Démonstration du théorème 7.3.2.

$$\boldsymbol{D}_{AB} = \begin{pmatrix} \sum_{t=1}^{n_{\mathrm{T}}} (x_{ti_1} - x_{tj_1})^2 & \dots & \sum_{t=1}^{n_{\mathrm{T}}} \left(x_{ti_1} - x_{tj_{m_{CB}}}\right)^2 \\ \vdots & \ddots & \vdots \\ \sum_{t=1}^{n_{\mathrm{T}}} \left(x_{ti_{m_{CA}}} - x_{tj_1}\right)^2 & \dots & \sum_{t=1}^{n_{\mathrm{T}}} \left(x_{ti_{m_{CA}}} - x_{tj_{m_{CB}}}\right)^2 \end{pmatrix}$$

$$\boldsymbol{D}_{AB} = \begin{pmatrix} \sum_{t=1}^{n_{\mathrm{T}}} \left(x_{ti_1}^2 + x_{tj_1}^2 - 2x_{ti_1}x_{tj_1}\right) & \dots & \sum_{t=1}^{n_{\mathrm{T}}} \left(x_{ti_1}^2 + x_{tj_{m_{CB}}}^2 - 2x_{ti_1}x_{tj_{m_{CB}}}\right) \\ \vdots & \ddots & \vdots \\ \sum_{t=1}^{n_{\mathrm{T}}} \left(x_{ti_{m_{CA}}}^2 + x_{tj_1}^2 - 2x_{ti_{m_{CA}}}x_{tj_1}\right) & \dots & \sum_{t=1}^{n_{\mathrm{T}}} \left(x_{ti_{m_{CA}}}^2 + x_{tj_{m_{CB}}}^2 - 2x_{ti_{m_{CA}}}x_{tj_{m_{CB}}}\right) \end{pmatrix}$$

$$\boldsymbol{D}_{AB} = \begin{pmatrix} \sum_{t=1}^{n_{\mathrm{T}}} x_{ti_1}^2 & \dots & \sum_{t=1}^{n_{\mathrm{T}}} x_{ti_1}^2 \\ \vdots & \ddots & \vdots \\ \sum_{t=1}^{n_{\mathrm{T}}} x_{ti_{m_{CA}}}^2 & \dots & \sum_{t=1}^{n_{\mathrm{T}}} x_{ti_{m_{CA}}}^2 \end{pmatrix} + \begin{pmatrix} \sum_{t=1}^{n_{\mathrm{T}}} x_{tj_1}^2 & \dots & \sum_{t=1}^{n_{\mathrm{T}}} x_{tj_{m_{CB}}}^2 \\ \vdots & \ddots & \vdots \\ \sum_{t=1}^{n_{\mathrm{T}}} x_{tj_1}^2 & \dots & \sum_{t=1}^{n_{\mathrm{T}}} x_{tj_{m_{CB}}}^2 \end{pmatrix}$$
$$- 2 \begin{pmatrix} \sum_{t=1}^{n_{\mathrm{T}}} x_{ti_1}x_{tj_1} & \dots & \sum_{t=1}^{n_{\mathrm{T}}} x_{ti_1}x_{tj_{m_{CB}}} \\ \vdots & \ddots & \vdots \\ \sum_{t=1}^{n_{\mathrm{T}}} x_{ti_{m_{CA}}}x_{tj_1} & \dots & \sum_{t=1}^{n_{\mathrm{T}}} x_{ti_{m_{CA}}}x_{tj_{m_{CB}}} \end{pmatrix}$$

$$\boldsymbol{D}_{AB} = \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} \times \begin{pmatrix} \sum_{t=1}^{n_{\mathrm{T}}} x_{ti_1}^2 & \dots & \sum_{t=1}^{n_{\mathrm{T}}} x_{ti_{m_{CA}}}^2 \end{pmatrix} + \begin{pmatrix} \sum_{t=1}^{n_{\mathrm{T}}} x_{tj_1}^2 \\ \vdots \\ \sum_{t=1}^{n_{\mathrm{T}}} x_{tj_{m_{CB}}}^2 \end{pmatrix} \times \begin{pmatrix} 1 & \dots & 1 \end{pmatrix}$$
$$- 2 \begin{pmatrix} x_{1i_1} & \dots & \boldsymbol{X}_{n_{\mathrm{T}}i_1} \\ \vdots & \ddots & \vdots \\ x_{1i_{m_{CA}}} & \dots & x_{n_{\mathrm{T}}i_{m_{CA}}} \end{pmatrix} \times \begin{pmatrix} x_{1j_1} & \dots & x_{1j_{m_{CB}}} \\ \vdots & \ddots & \vdots \\ x_{n_{\mathrm{T}}j_1} & \dots & x_{n_{\mathrm{T}}j_{m_{CB}}} \end{pmatrix}$$

$$\boldsymbol{D}_{AB} = \mathbf{1}_{\boldsymbol{m_{CB}}}\boldsymbol{s}_B + \boldsymbol{s}_A^\top \mathbf{1}_{\boldsymbol{m_{CA}}}^\top - 2\boldsymbol{A}^\top\boldsymbol{B}$$

□

7.3.2.2 Implémentation

Au début du programme, le processus CPU va copier le vecteur $\mathbf{1}$ de la mémoire CPU vers la mémoire GPU À chaque étape de la boucle externe, le processus sur CPU initialise la tuile $\boldsymbol{A}$ et calcule $\boldsymbol{s}_A$. Ensuite, $\boldsymbol{s}_A$ et $\boldsymbol{A}$ sont copiés depuis le CPU vers le GPU. Le calcul de $\boldsymbol{D}_A$ est effectué en trois étapes toutes sur le GPU :

1. $\boldsymbol{D}_A = \mathbf{1}_{\boldsymbol{m_{CA}}}\boldsymbol{s}_A$
2. $\boldsymbol{D}_A = \boldsymbol{D}_A + \boldsymbol{s}_A^\top \mathbf{1}_{\boldsymbol{m_{CA}}}^\top$
3. $\boldsymbol{D}_A = \boldsymbol{D}_A - 2\boldsymbol{A}^\top\boldsymbol{A}$

La matrice $\boldsymbol{D}_A$ est ensuite copiée du GPU vers le CPU. À chaque étape de la boucle interne, le processus CPU initialise $\boldsymbol{B}$ et calcule $\boldsymbol{s}_B$. Puis $\boldsymbol{s}_B$ et $\boldsymbol{B}$ sont copiés du CPU vers le GPU. Le calcul de $\boldsymbol{D}_{AB}$ s'effectue ensuite en trois étapes toutes sur le GPU :

1. $\boldsymbol{D}_{AB} = \mathbf{1}_{\boldsymbol{m_{CA}}}\boldsymbol{s}_B$
2. $\boldsymbol{D}_{AB} = \boldsymbol{D}_{AB} + \boldsymbol{s}_A^\top \mathbf{1}_{\boldsymbol{m_{CB}}}^\top$
3. $\boldsymbol{D}_{AB} = \boldsymbol{D}_{AB} - 2\boldsymbol{A}^\top\boldsymbol{B}$

La matrice $\boldsymbol{D}_{AB}$ est ensuite copiée du GPU vers le CPU.

Les appels aux noyaux GPU sont asynchrones, c'est-à-dire qu'une fois le noyau lancé, le CPU reprend la main. Le CPU peut donc effectuer des tâches pendant que le GPU calcule. Dans notre implémentation, le CPU copie les matrices de distance dans le vecteur de distances pendant que le GPU calcule. La boucle externe est décroissante. Ainsi, la distance entre les colonnes de la tuile $\boldsymbol{A}_{n_{CB}}$ est écrite lors du calcul de la distance entre les colonnes de la tuile $\boldsymbol{A}_{n_{CB}-1}$. La distance entre les colonnes de la tuile $\boldsymbol{A}_i$ pour $i \neq n_{CB}$ est écrite durant le calcul de la distance entre la tuile $\boldsymbol{A}_i$ et la tuile $\boldsymbol{A}_{n_{CB}}$. La distance entre les colonnes de la tuile $\boldsymbol{A}_i$ et la tuile $\boldsymbol{X}_{\cdot j}$, $j \geq i$ est écrite pendant le calcul de la distance entre la tuile $\boldsymbol{A}_i$ et la tuile $\boldsymbol{A}_{j+1}$.

Sur certains GPUs, on peut superposer l'exécution d'un noyau et la copie du CPU vers le GPU ou du GPU vers le CPU en utilisant des *streams*. Pour cela, les copies doivent être asynchrones pour que le CPU puisse lancer le noyau GPU après avoir lancé la copie. En reprenant le calcul de $\boldsymbol{D}_A$, on s'aperçoit qu'une fois le vecteur $\boldsymbol{s}_A$ copié sur le GPU, les deux premiers calculs peuvent être lancés, même si la tuile $\boldsymbol{A}$ n'est pas encore copiée sur le GPU. Pour cela, on utilise deux *streams* : sur le premier *stream* on lance la copie de $\boldsymbol{s}_A$ puis les deux premiers calculs nécessaires pour $\boldsymbol{D}_A$ et le *stream* 2 copie $\boldsymbol{A}$; le *stream* 1 attend ensuite que le *stream* 2 ait fini la copie pour lancer le dernier calcul. Le *stream* 2 est chargé de la copie de $\boldsymbol{D}_A$. Le CPU doit donc se synchroniser avec le *stream* 2 pour copier $\boldsymbol{D}_A$ dans le vecteur de distances. Pour le calcul de $\boldsymbol{D}_{AB}$, on utilise aussi 2 *streams* : le *stream* 1 copie $\boldsymbol{s}_B$ puis calcule les deux premières étapes pendant que le *stream* 2 copie la tuile $\boldsymbol{B}$. Une fois la copie de $\boldsymbol{B}$ terminée, le *stream* 1 achève le calcul de $\boldsymbol{D}_{AB}$. Le calcul achevé, le *stream* 2 copie $\boldsymbol{D}_{AB}$ sur le CPU. Le calcul est illustré à la figure 7.18 pour un découpage de la matrice de données en deux tuiles tel que :

$$X = \begin{pmatrix} \boldsymbol{X}_{\cdot 1} & \boldsymbol{X}_{\cdot 2} \end{pmatrix} \quad \boldsymbol{D} = \begin{pmatrix} \boldsymbol{D}(\boldsymbol{X}_{\cdot 1}, \boldsymbol{X}_{\cdot 1}) & \boldsymbol{D}(\boldsymbol{X}_{\cdot 1}, \boldsymbol{X}_{\cdot 2}) \\ \boldsymbol{D}(\boldsymbol{X}_{\cdot 2}, \boldsymbol{X}_{\cdot 1}) & \boldsymbol{D}(\boldsymbol{X}_{\cdot 2}, \boldsymbol{X}_{\cdot 2}) \end{pmatrix}$$

7.4 $2^{ème}$ version de la parallélisation

La première version de la parallélisation avec OpenMP n'est pas satisfaisante. En effet, les accélérations obtenues sont loin de s'approcher d'une accélération linéaire.

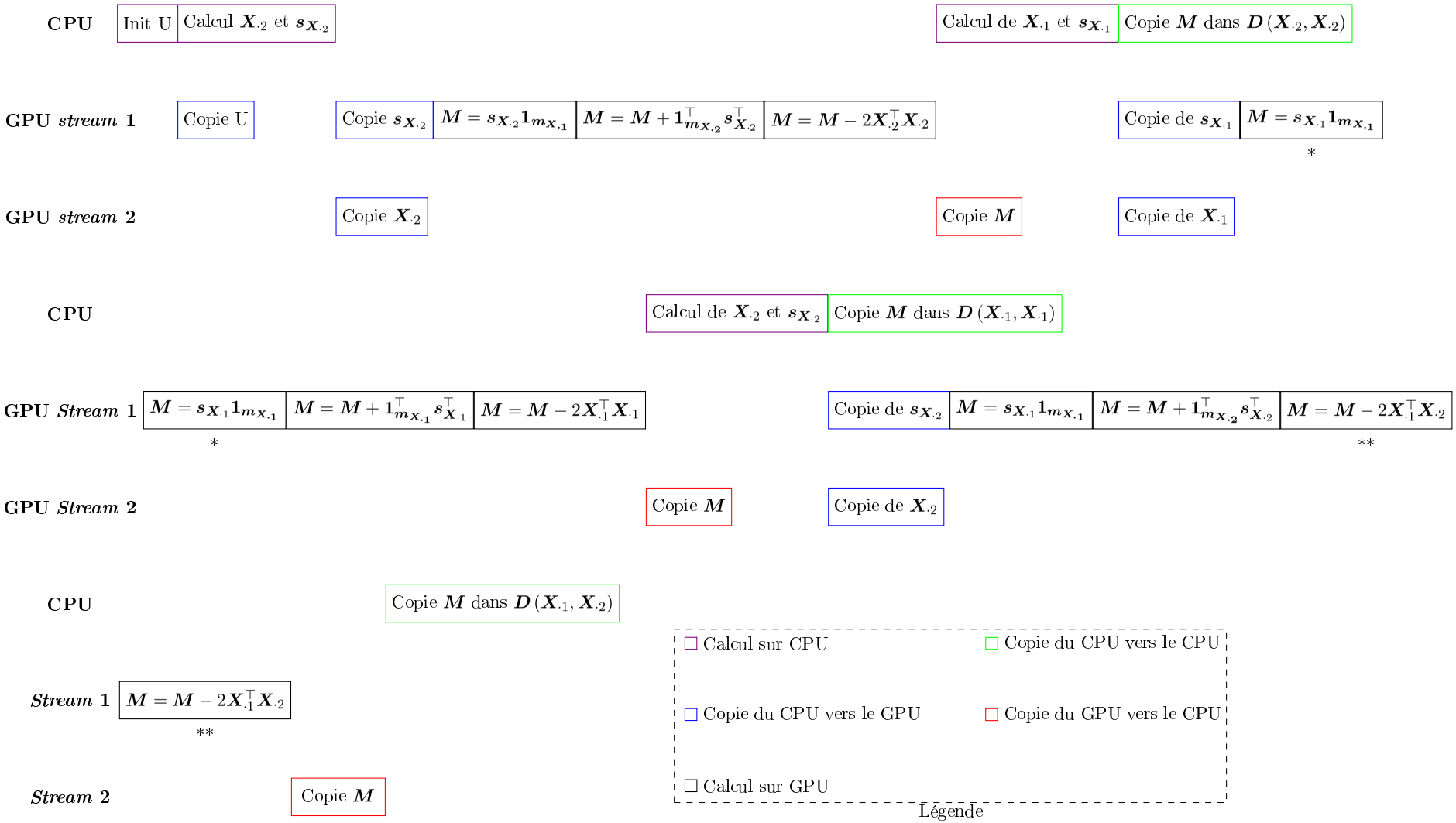


FIGURE 7.18 – Version 1-Calcul de la distance sur GPU : illustration pour deux tuiles.

7.4.1 Parallélisation avec OpenMP

Le calcul de la distance décrit en section 7.3.1 présente un défaut : il y a un processus léger qui est chargé de la première ligne de la matrice de distance. Un seul processus léger est donc en charge de calculer la distance entre la tuile $\boldsymbol{X}_{\cdot 1}$ et $\boldsymbol{X}_{\cdot j}$ pour j de 2 à n_{CB}. Le calcul de la distance a donc été amélioré. Cette fois-ci au lieu de répartir les lignes de tuiles de la matrice de distances, on répartit les tuiles. Une telle répartition est illustrée en figure 7.19.

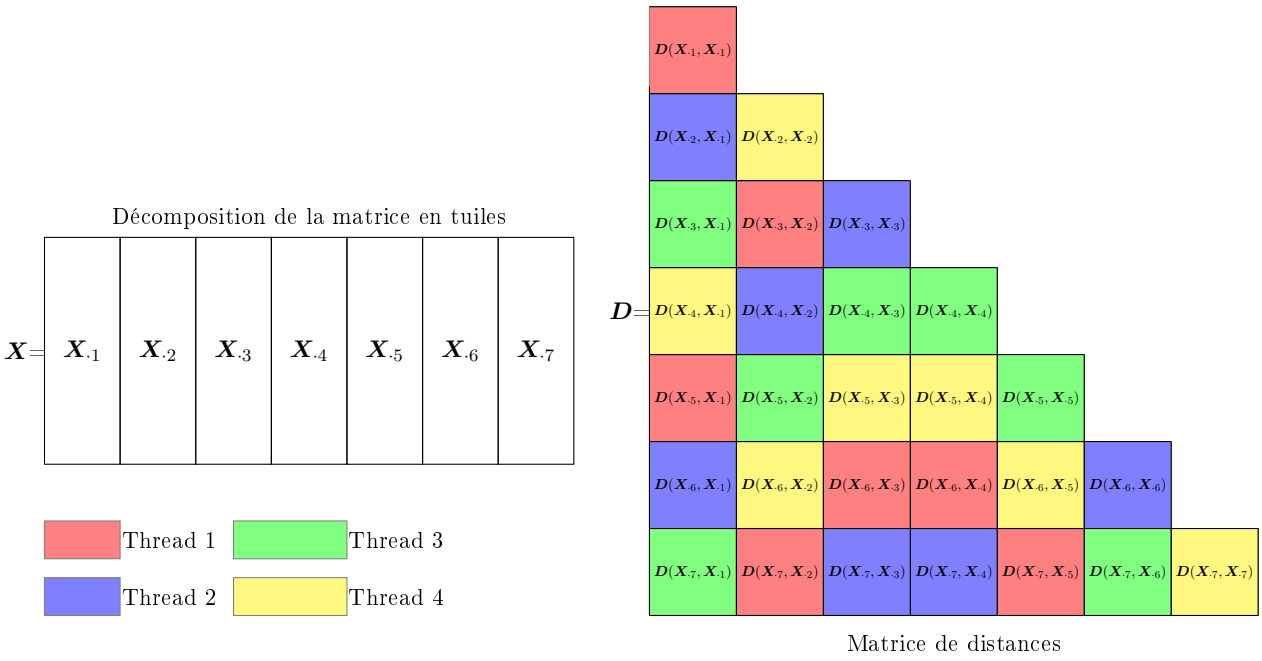


FIGURE 7.19 – 2ème version : Exemple de calcul de la distance avec OpenMP pour 7 tuiles et 4 processus .

Cependant, contrairement à la figure, lorsqu'on distribue une tuile de la matrice de distances sur un processus léger, ce processus ne connaît pas entre quelles tuiles de la matrice d'entrée $\boldsymbol{X}$, il doit calculer la distance. Pour cela, il faut retrouver à partir de l'indice k de la tuile de distance les indices (i, j) des tuiles de la matrice $\boldsymbol{X}$. Si on note $\boldsymbol{D}_k$ la $k^{ème}$ tuile de la matrice de distances, on a $\boldsymbol{D}_k = \boldsymbol{D}(\boldsymbol{X}_{\cdot i}, \boldsymbol{X}_{\cdot j})$. La figure 7.20 illustre une telle correspondance. Contrairement au vecteur de distances, le découpage en tuiles de la matrice de distances a été indexé en considérant la matrice triangulaire inférieure. La section 7.2.2 établit la correspondance entre l'indice k et les indices (i, j). Pour un indice $k \in \{1, \ldots, n_{CB}\}$, on a

$$i = k + \frac{n_{CB}(1 - n_{CB})}{2} + \frac{1}{2}\left\lfloor \frac{\sqrt{8\left(\frac{n_{CB}(n_{CB}+1)}{2} - k\right) + 1} - 1}{2} \right\rfloor \left(\left\lfloor \frac{\sqrt{8\left(\frac{n_{CB}(n_{CB}+1)}{2} - k\right) + 1} - 1}{2} \right\rfloor + 1 \right)$$

$$j = n_{CB} - \left\lfloor \frac{\sqrt{8\left(\frac{n_{CB}(n_{CB}+1)}{2} - k\right) + 1} - 1}{2} \right\rfloor$$

7.4.2 Parallélisation avec CUDA

L'un des inconvénients de la première méthode présentée est que peu de noyaux CUDA s'exécutent en parallèle sur le GPU. Comme illustré sur la figure 7.18, la première version ne superpose que des copies ou une copie avec un calcul. Mais on ne superpose pas plusieurs calculs.

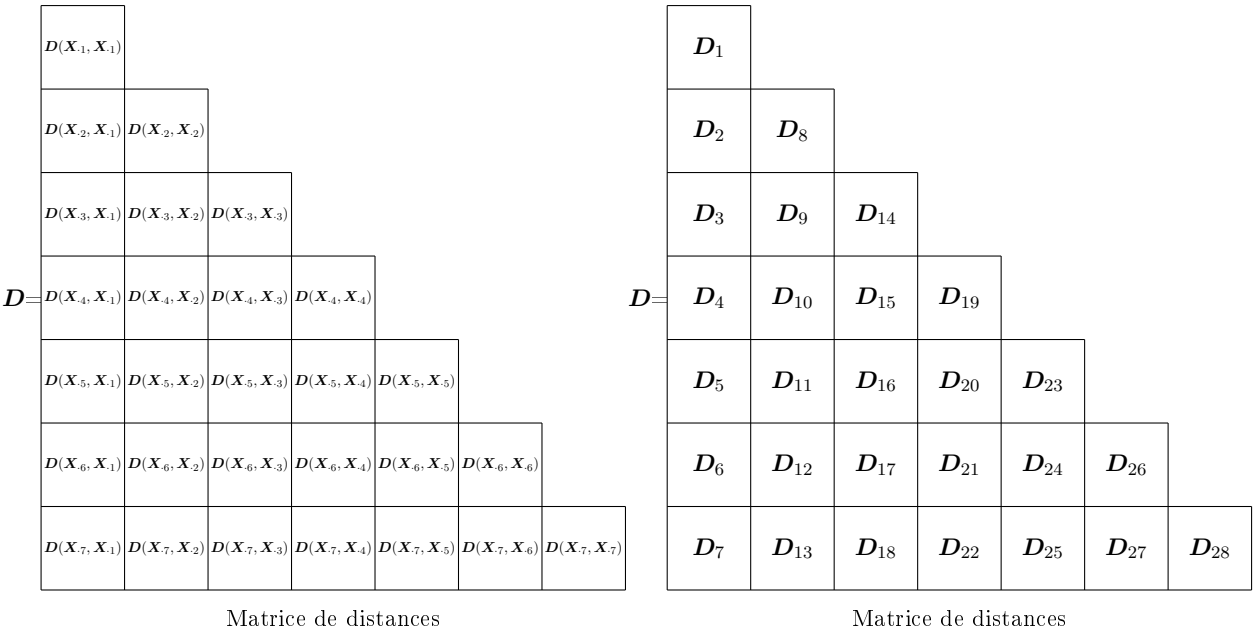


FIGURE 7.20 – $2^{ème}$ version : Correspondance entre l'indice des tuiles de distance et les tuiles de la matrice $\boldsymbol{X}$.

Plutôt que de parcourir les tuiles $\boldsymbol{X}_{.i}$ de la matrice $\boldsymbol{X}$, l'idée est de parcourir les tuiles $\boldsymbol{D}_k$ de la matrice de distances. Lorsqu'on traite la tuile $\boldsymbol{D}_k$, il faut trouver les indices (i, j) tels que $\boldsymbol{D}_k = \boldsymbol{D}(\boldsymbol{X}_{.i}, \boldsymbol{X}_{.j})$ et copier les tuiles $\boldsymbol{X}_{.i}$ et $\boldsymbol{X}_{.j}$ sur le GPU. On peut ensuite calculer $\boldsymbol{s}_{\boldsymbol{X}_{.i}}$ et $\boldsymbol{s}_{\boldsymbol{X}_{.j}}$ puis calculer $\boldsymbol{M} = \boldsymbol{D}(\boldsymbol{X}_{.i}, \boldsymbol{X}_{.j})$. Bien qu'on puisse calculer $\boldsymbol{s}_{\boldsymbol{X}_{.i}}$ et $\boldsymbol{s}_{\boldsymbol{X}_{.j}}$ de manière indépendante et donc en même temps sur le GPU, le calcul de $\boldsymbol{M}$ nécessite trois appels dépendants à *General Matrix Multiplication* (GEMM). Pour augmenter la concurrence entre les noyaux CUDA, il faudrait lancer plusieurs GEMMs indépendantes. Ceci est possible, si on calcule $\boldsymbol{D}_k$ pour plusieurs k. Pour cela, il faut dérouler la boucle sur k c'est-à-dire traiter plusieurs k pour une même itération. Prenons le cas simple où on découpe la matrice $\boldsymbol{X}$ en quatre tuiles. Dans ce cas, la matrice de distances est composée de seize tuiles, mais on ne calcule grâce à la symétrie que dix tuiles. Si on ne déroule pas la boucle, on va calculer $\boldsymbol{D}_1$, puis $\boldsymbol{D}_2$ jusqu'à calculer $\boldsymbol{D}_{10}$. Déroulons maintenant la boucle avec une profondeur de 2 (c'est-à-dire qu'on traite deux tuiles à la fois). Cette fois-ci on va calculer à la première itération $\boldsymbol{D}_1$ et $\boldsymbol{D}_2$, puis à la deuxième itération $\boldsymbol{D}_3$ et $\boldsymbol{D}_4$ jusqu'à calculer à la cinquième itération $\boldsymbol{D}_9$ et $\boldsymbol{D}_{10}$. La figure 7.21 illustre le procédé pour une profondeur de 3.

L'intérêt de dérouler la boucle sur les k est qu'en créant un nombre de *streams* égal à la profondeur du déroulement, on peut lancer sur chaque *stream* le calcul d'une tuile $\boldsymbol{D}_k$ de manière indépendante. La contrepartie au déroulement de la boucle est qu'il faut allouer plus de mémoire sur CPU et sur GPU. Pour effectuer le calcul de la distance $\boldsymbol{D}_k = \boldsymbol{D}(\boldsymbol{X}_{.i}, \boldsymbol{X}_{.j})$, chaque *stream* doit en effet disposer des tuiles $\boldsymbol{X}_{.i}$ et $\boldsymbol{X}_{.j}$ initialisées sur CPU et copiées sur le GPU, des vecteurs $\boldsymbol{s}_{\boldsymbol{X}_{.i}}$ et $\boldsymbol{s}_{\boldsymbol{X}_{.j}}$ calculés sur GPU et de la matrice $\boldsymbol{M}$ où stocker la distance calculée sur GPU puis copiée sur CPU. Pour une profondeur p de boucle, il faut p fois plus de mémoire. La profondeur choisie dépend à la fois de la taille de tuiles choisies puisqu'il faut disposer d'assez de mémoire sur GPU et également du nombre d'unités de calcul disponibles pour la précision choisie. La figure 7.22 illustre le calcul de la distance sur GPU pour un découpage de la matrice $\boldsymbol{X}$ en deux tuiles, une profondeur de boucle égale à 2 et donc l'utilisation de deux *streams*. La figure montre bien que plusieurs noyaux de calcul sont lancés simultanément sur GPU. Cependant, ces calculs sont décalés car même si on déroule la boucle, il faut que le CPU ait eu le temps de construire les tuiles avant de lancer le calcul sur GPU. De plus, le CPU est inactif pendant que le GPU calcule. Le CPU pourrait prendre en charge le calcul d'une tuile $\boldsymbol{D}_k$ pendant que le GPU calcule.

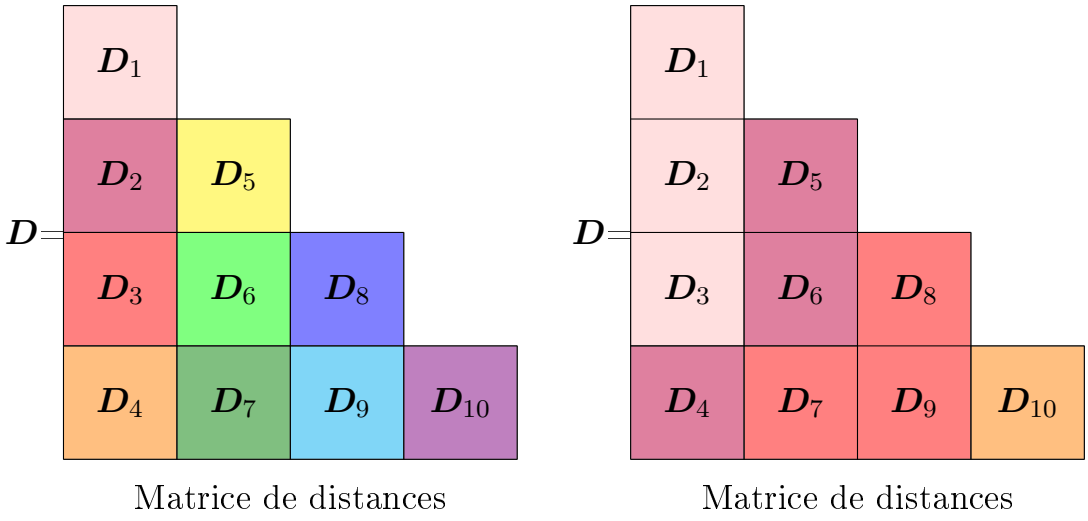


FIGURE 7.21 – 2ème version : Calcul de la distance sans déroulement de boucle (à gauche) et avec déroulement de la boucle (à droite) pour une profondeur de 3. Chaque couleur représente une itération : les tuiles d'une même couleur sont traitées durant la même itération.

Cependant, le temps de calcul sur CPU est largement supérieur au temps de calcul sur GPU. C'est pourquoi il n'y a pas de superposition entre calcul sur CPU et calcul sur GPU.

7.4.3 Parallélisation avec CUDA+OpenMP

Nous proposons ici d'utiliser à la fois OpenMP et CUDA. Les calculs sont faits sur le GPU mais on se sert de OpenMP pour mieux superposer les calculs sur le GPU.

Reprenons la deuxième version du calcul de la distance (**cf. Figure 7.22**). Ce qui est frappant est que les calculs sur GPU ne débutent pas en même temps alors qu'ils sont indépendants. Ce décalage est dû au fait que l'initialisation des tuiles se fait séquentiellement sur le CPU. L'idée est d'utiliser OpenMP pour initialiser les tuiles simultanément et donc lancer les calculs sur GPU en même temps. La figure 7.23 illustre le procédé pour un découpage de la matrice avec deux tuiles, un déroulement de boucle de profondeur 2 et l'utilisation de deux processus légers et de deux *streams*.

7.5 3ème version de la parallélisation

7.5.1 Calcul de la distance par découpage des lignes et des colonnes

La taille de la tuile change les performances du calcul de la distance. Afin d'obtenir les meilleures performances, l'idée est donc de découper la matrice $\boldsymbol{X}$ de données à la fois en tuiles de colonnes et en blocs de lignes. Si les tuiles $\boldsymbol{X}_{IJ}$ sont suffisamment petites, elles peuvent entrer dans la mémoire cache du processus léger qui va la traiter. Si on reprend l'expression de la distance euclidienne entre deux vecteurs $\boldsymbol{x}_{j_1}$ et $\boldsymbol{x}_{j_2}$, on se rend compte qu'on peut la décomposer sur chaque bloc de lignes $\mathcal{I}_k$:

$$\|\boldsymbol{x}_{j_1} - \boldsymbol{x}_{j_2}\|^2 = \sum_{i=1}^{n_\mathrm{T}} (x_{ij_1} - x_{ij_2})^2 = \sum_{k=1}^{m_R} \sum_{i \in \mathcal{I}_k} (\boldsymbol{x}_{ij_1} - \boldsymbol{x}_{ij_2})^2 \ .$$

L'idée est donc pour deux tuiles $\boldsymbol{X}_{.J_1}$ et $\boldsymbol{X}_{.J_2}$ de charger la ligne $\mathcal{I}_i$, de calculer la distance sur ces lignes et d'ajouter cette distance calculée à $\boldsymbol{D}_{J_1 J_2}$. On ajoute donc la distance sur les blocs de lignes à la distance déjà calculée $\mathcal{I}_I$ au fur et à mesure. Reste le problème du premier bloc de lignes $\mathcal{I}_1$ qu'on ne peut pas ajouter à la distance déjà calculée. Une manière serait d'initialiser le

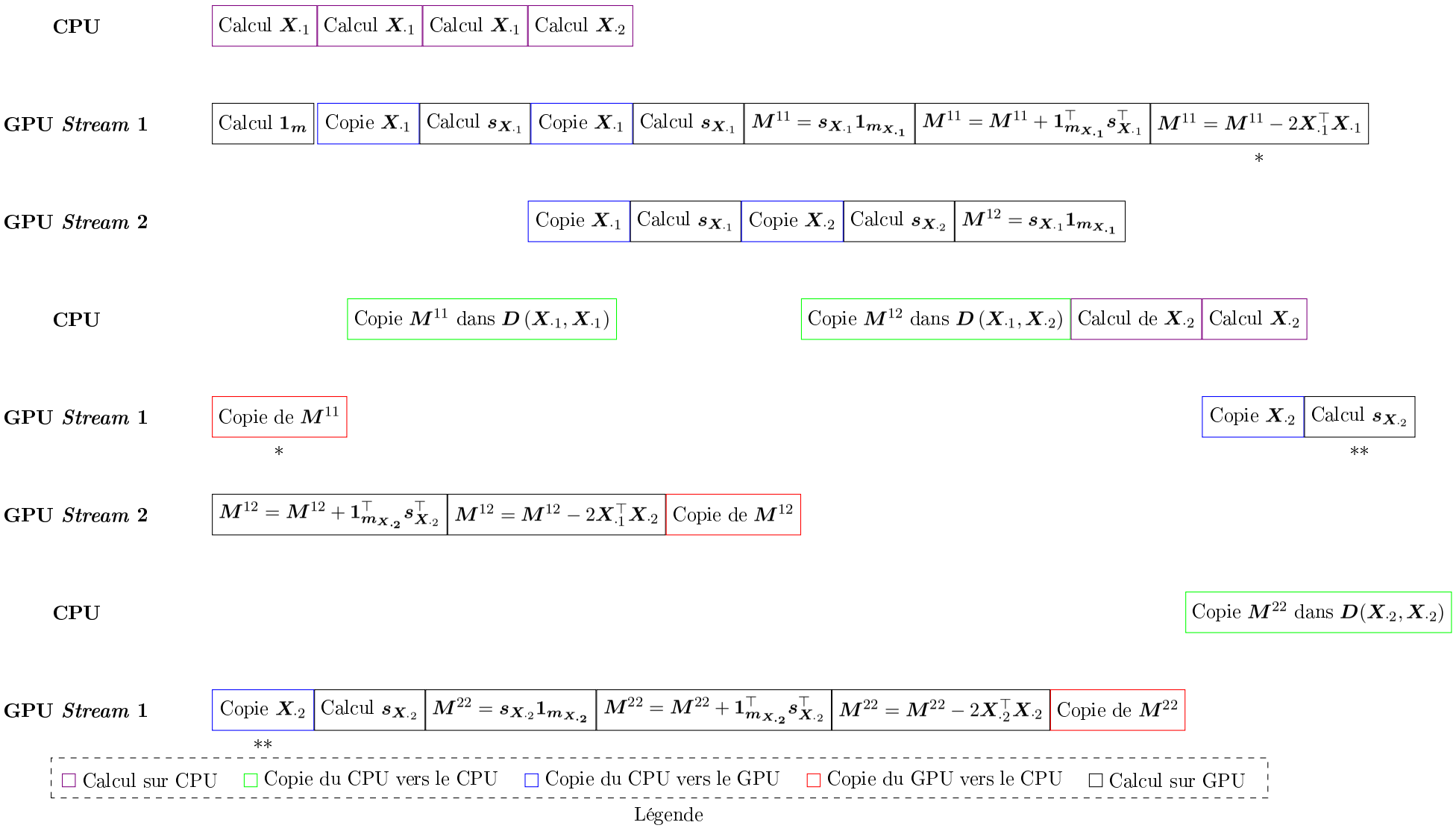


FIGURE 7.22 – Version 2-Calcul de la distance sur GPU : illustration pour deux tuiles.

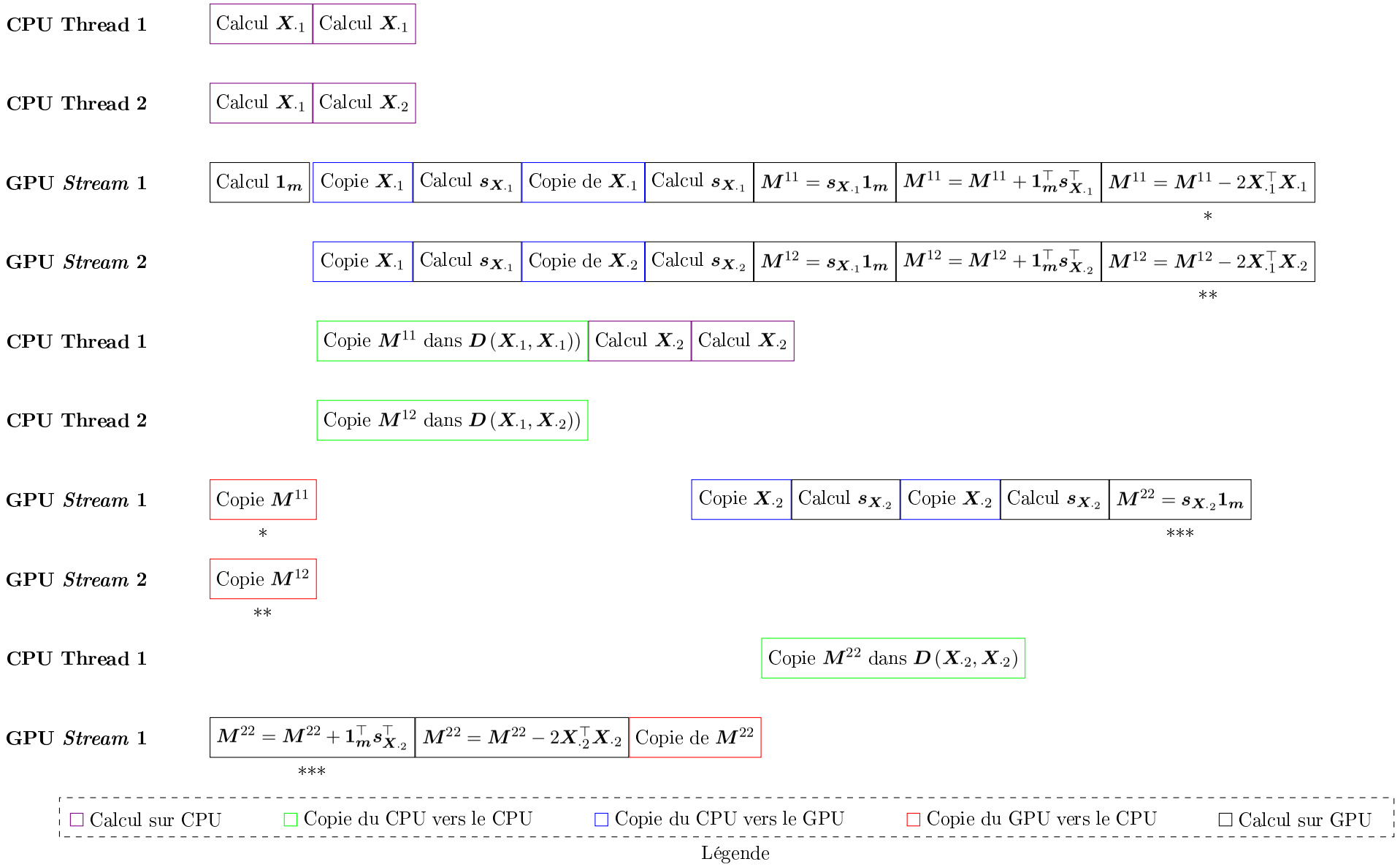


FIGURE 7.23 – Version 2 : Calcul de la distance avec OpenMP et CUDA .

vecteur de distances à zéro et on pourrait alors ajouter la distance sur le premier bloc de lignes au vecteur de distances. Cependant, le vecteur de distances est de taille $\frac{n_V(n_V-1)}{2}$ donc l'initialiser à zéro devient coûteux lorsque n_V est grand. L'initialisation se fera donc avec le premier bloc de lignes $\mathcal{I}_1$ puisqu'il faut le calculer dans tous les cas. En effet, même si $m_R >= n_T$, on a au moins un bloc de lignes $\mathcal{I}_1$ éventuellement de taille plus petite que la taille m_R prévue. Si $n_T > m_R$, on a alors au moins deux blocs de lignes à parcourir.

L'algorithme 12 montre le calcul de la distance avec un découpage des lignes et des colonnes. La ligne 1 correspond au calcul du nombre n_{CB} de tuiles nécessaires pour couvrir toutes les colonnes, du nombre n_{RB} de blocs de lignes pour couvrir toutes les lignes qui correspond aussi aux nombres d'itérations qu'on va faire ; et du nombre de blocs n_B de la matrice de distances $\boldsymbol{D}$. On dispose de n_{CB} tuiles pour lesquelles il faut calculer la distance pour toutes les paires en comptant les paires de tuiles identiques (tuiles sur la diagonale) d'où la formule utilisée pour n_B. La ligne 2 correspond à l'initialisation du premier bloc de lignes : $\mathcal{I}_1$ l'indice m_1 correspond à la première ligne dans le bloc et m_2 correspond à la dernière ligne du bloc $\mathcal{I}_1$. La boucle commençant à la ligne 3 correspond au parcours des blocs de la matrice de distances. Pour une tuile $\boldsymbol{D}_\kappa$, il faut trouver à quelles tuiles ce bloc correspond : le calcul utilise l'expression de ψ_3 démontrée en section 7.2.4 (ligne 4). Une fois les tuiles $\boldsymbol{X}_{.K}$ et $\boldsymbol{X}_{.L}$ identifiées, on détermine la colonne de début (i_1 et j_1) et la colonne de fin (i_2 et j_2) des tuiles en ligne 5. On distincte ensuite le cas $K > L$ à la ligne 6 qui correspond à une tuile de $\boldsymbol{D}$ en dehors de la diagonale et le cas $K \leq L$ correspondant à une tuile diagonale en ligne 12. Dans le cas $K > L$, il faut calculer la distance sur toutes les paires de colonnes (i, j) avec $i \in \mathcal{J}_K$ et $j \in \mathcal{J}_L$. Cette distance est calculée par la boucle 10 avec une initialisation de la distance à la ligne 9 puisqu'on travaille dans le premier bloc de lignes $\mathcal{I}_1$. Dans le cas K $\leq$ L, la tuile de $\boldsymbol{D}$ considérée est une tuile diagonale, il ne faut donc calculer que la partie triangulaire supérieure stricte (sans la diagonale). C'est pourquoi la boucle 14 est dépendante de la boucle 13. La distance est ensuite calculée par l'initialisation 15 et la boucle 16. L'indice où stocker la distance est donné par k en ligne 9 et 15 selon la fonction ψ_2 donnée en section 7.2.3. La boucle 18 permet de calculer la distance sur les autres blocs de lignes $\mathcal{I}_i$. La boucle interne 20 est donc la même que la boucle 3 excepté que les valeurs de m_1 et m_2 sont différentes. La seule différence entre les boucles ligne 3 et 20 est le calcul de la distance qui en ligne 27 et 33 n'a pas besoin d'être initialisée puisqu'elle a déjà été partiellement calculée lors les boucles 9 et 15. Le calcul de la distance est ici montré sous forme de boucle (**cf. Lignes 9, 9, 27 et 33**) mais en pratique, il est remplacé par l'utilisation du calcul matriciel ou par l'emploi de la fonction sum ou dot_product.

L'algorithme 12 a cependant un défaut. En effet, en faisant d'abord la boucle sur les itérations, pour calculer la distance $\boldsymbol{D}_{KL}$, la tuile $\boldsymbol{D}_{KL}$ va être chargée en mémoire à chaque accès donc n_{RB} fois. De plus lorsqu'on calcule la distance sur les m_R lignes du bloc $\mathcal{I}_i$, on ne charge pas seulement ces m_R lignes des tuiles $\boldsymbol{X}_{.K}$ et $\boldsymbol{X}_{.L}$ mais aussi les lignes suivantes si cela rentre dans la mémoire cache. Supposons qu'au moment de calculer la distance sur le bloc de lignes $\mathcal{I}_i$, on charge en mémoire le $\mathcal{I}_i$ et le $\mathcal{I}_{i+1}$, ce bloc $\mathcal{I}_{i+1}$ occupe de l'espace mémoire mais il n'est pas utilisé. Lorsqu'on passe au calcul sur le bloc $\mathcal{I}_{i+1}$, ces lignes précédemment chargées ne sont plus disponibles en mémoire car on a chargé entre-temps les lignes de bloc $\mathcal{I}_i$ pour les autres tuiles. On ne respecte donc pas la localité spatiale des données. Inversons l'ordre des boucles, c'est-à-dire que la boucle sur les blocs $\mathcal{I}_i$ des lignes est interne à la boucle sur les tuiles de la matrice de distances. Après avoir calculé la distance sur le bloc de lignes $\mathcal{I}_i$, on calcule la distance sur le bloc de lignes $\mathcal{I}_{i+1}$. Si en chargeant le $\mathcal{I}_i$, le $\mathcal{I}_{i+1}$ est aussi chargé en mémoire, le calcul de la distance sur le bloc $\mathcal{I}_{i+1}$ n'a pas besoin de charger les lignes depuis la mémoire centrale car elles sont présentes en mémoire cache. Éviter un chargement des données depuis la mémoire cache permet de gagner du temps.

lorsqu'on calcule la distance sur les lignes du bloc $\mathcal{I}_{i+1}$: elles sont déjà chargées en mémoire cache ce qui évite d'avoir à les charger depuis la mémoire vive et donc de gagner du temps.

Lorsqu'on calcule la distance pour chaque paire de tuiles, on s'aperçoit que certains cal-

Algorithme 12 : Calcul de la distance par découpage en tuiles de colonnes et en blocs de lignes.

Entrée : $\boldsymbol{X}$ matrice de taille $n_{\mathrm{T}} \times n_{\mathrm{V}}$; m_R nombre de lignes par bloc ; m_C nombre de colonnes par tuile
Sortie : $\boldsymbol{D}$ vecteur de taille $\frac{n_{\mathrm{V}}(n_{\mathrm{V}}-1)}{2}$

```
1  n_CB = ⌈n_V/m_C⌉ ; n_RB = ⌈n_T/m_R⌉ ; n_B = n_CB[n_CB+1)/2 ;
2  iter=1 ; m_1=1+(iter-1)m_R ; m_2=iter × m_R;
3  pour κ de 1 à n_B faire
4      (K, L) = ψ_3(κ) ;
5      i_1=1+(K-1)m_C ; i_2=min(Km_C, n_V) ; j_1=1+(L-1)m_C ; j_2=min(Lm_C, n_V) ;
6      si K >L alors
7          pour i de i_1 à i_2 faire
8              pour j de j_1 à j_2 faire
9                  k = j + (i-1)(i-2)/2 ; d(k)=(x_{m_1 i} − x_{m_2 j})^2;
10                 pour m de m_1 + 1 à m_2 faire
11                     d(k)=d(k)+(x_mi − x_mj)^2;
12     sinon
13         pour j de j_1 à j_2 faire
14             pour i de i_1 à j − 1 faire
15                 k = i + (j-1)(j-2)/2 ; d(k)=(x_{m_1 i} − x_{m_2 j})^2;
16                 pour m de m_1 + 1 à m_2 faire
17                     d(k)=d(k)+(x_mi − x_mj)^2;
18 pour iter de 2 à n_RB faire
19     m_1=1+(iter-1)m_R ; m_2=iter × m_R;
20     pour κ de 1 à n_B faire
21         (K, L) = ψ_3(κ);
22         i_1=1+(K-1)m_C ; i_2=min(Km_C, n_V) ; j_1=1+(L-1)m_C ; j_2=min(Lm_C, n_V);
23         si K >L alors
24             pour i de i_1 à i_2 faire
25                 pour j de j_1 à j_2 faire
26                     k = j + (i-1)(i-2)/2;
27                     pour m de m_1 à m_2 faire
28                         d(k)=d(k)+(x_mi − x_mj)^2;
29         sinon
30             pour j de j_1 à j_2 faire
31                 pour i de i_1 à j − 1 faire
32                     k = i + (j-1)(j-2)/2;
33                     pour m de m_1 à m_2 faire
34                         d(k)=d(k)+(x_mi − x_mj)^2;
```

culs sont faits plusieurs fois alors qu'ils pourraient être calculés une seule fois puis réutilisés. Développons l'expression du calcul de la distance euclidienne :

$$\begin{aligned}\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2 &= \sum_{k=1}^{n_\mathrm{T}} (x_{ki} - x_{kj})^2 \\ &= \sum_{k=1}^{n_\mathrm{T}} (x_{ki}^2 + x_{kj}^2 - 2x_{ki}x_{kj}) \\ &= \sum_{k=1}^{n_\mathrm{T}} x_{ki}^2 + \sum_{k=1}^{n_\mathrm{T}} x_{kj}^2 - 2\sum_{k=1}^{n_\mathrm{T}} x_{ki}x_{kj} \\ &= \|\boldsymbol{x}_j\|^2 + \|\boldsymbol{x}_i\|^2 - 2\sum_{k=1}^{n_\mathrm{T}} x_{ki}x_{kj}\end{aligned} \quad . \tag{7.8}$$

Pour chaque colonne $\boldsymbol{x}_j$ de la matrice $\boldsymbol{X}$, sa norme $\|\boldsymbol{x}_j\|$ est donc utilisée dans le calcul de la distance entre $\boldsymbol{x}_j$ et $\boldsymbol{x}_i$ avec $j \leq i \leq n_\mathrm{V}$. On peut donc calculer la norme de chaque colonne à l'avance pour la réutiliser par la suite dans le calcul de la distance. Ce calcul de la norme peut se faire en utilisant le découpage des n_T lignes en blocs.

L'algorithme 13 reprend l'algorithme 12 en inversant l'ordre des boucles et en séparant le calcul de la norme des colonnes. Cet algorithme commence aussi par calculer le nombre de tuiles pour couvrir toutes les colonnes, le nombre de blocs de lignes pour couvrir toutes les lignes et le nombre de blocs de la matrice de distances en ligne 1. La boucle commençant en ligne 2 calcule la norme pour chaque colonne $\boldsymbol{x}_j$ de la matrice $\boldsymbol{X}$ en itérant le calcul sur les blocs de lignes. La ligne 3 calcule les indices de la ligne de début m_1 et de la ligne de fin m_2 du bloc de lignes $\mathcal{I}_1$. La distance est initialisée avec la valeur de $\boldsymbol{x}_j$ à la ligne m_1 en ligne 3 puis les autres lignes sont ajoutées avec la boucle en ligne 4. Les blocs de lignes suivants sont traités par la boucle en ligne 6 avec calcul des indices de ligne de début et de fin en ligne 7 puis la boucle sur les lignes en ligne 8. La boucle commençant en ligne 10 parcourt les blocs de la matrice de distances. Pour chaque tuile de la matrice de distances, on initialise les indices de ligne m_1 et m_2 du bloc $\mathcal{I}_1$ en ligne 11 et on trouve à quelles tuiles de $\boldsymbol{X}$ la tuile correspond en ligne 12 puis on initialise les indices de colonnes des tuiles $\boldsymbol{X}_{\cdot K}$ et $\boldsymbol{X}_{\cdot L}$ en ligne 13 . On distingue encore le cas des tuiles de $\boldsymbol{D}$ hors diagonale (boucle ligne 14) et le cas des tuiles sur la diagonale (boucle ligne 20). La différence avec l'algorithme 12 est que la distance est ici initialisée avec la somme des normes des colonnes $\boldsymbol{x}_i$ et $\boldsymbol{x}_j$ en lignes 17 et 23 puis les boucles en lignes 18 et 24 calculent uniquement le double produit des termes. Et les calculs sont recommencés pour les autres blocs de lignes grâce à la boucle 26.

7.5.2 Parallélisations

7.5.2.1 Parallélisation avec OpenMP

Sont parallélisées avec OpenMP les boucles ligne 2 et 10 de l'algorithme 13. Autrement dit, chaque processus léger prend en charge une colonne $\boldsymbol{x}_j$ de la matrice et calcule sa norme, quand le calcul est terminé, il prend en charge une autre colonne et ce jusqu'à ce que toutes les normes aient été calculées. Les tuiles de la matrice de distances sont ensuite réparties dynamiquement entre les processus légers. Chaque processus léger a donc en charge une paire de tuiles de la matrice $\boldsymbol{X}$ et calcule la distance en découpant la tuile en blocs de lignes. Lorsque le calcul de la distance est fini, le processus léger prend en charge une autre paire de tuiles. La répartition dynamique entre les processus légers est importante car le temps de calcul est différent si la tuile $\boldsymbol{D}_\kappa$ est une tuile sur la diagonale par rapport à une tuile non diagonale car pour une tuile diagonale, par symétrie de la matrice de distances, on ne calcule que la partie triangulaire inférieure stricte.

Algorithme 13 : Calcul de la distance par découpage en tuiles de colonnes et en blocs de lignes (amélioré) .

Entrée : $\boldsymbol{X}$ matrice de taille $n_{\mathrm{T}} \times n_{\mathrm{V}}$; m_R nombre de lignes par bloc ; m_C nombre de colonnes par tuile

Sortie : $\boldsymbol{D}$ vecteur de taille $\frac{n_{\mathrm{V}}(n_{\mathrm{V}}-1)}{2}$

```
1  n_CB = ⌈n_V/m_C⌉ ; n_RB = ⌈n_T/m_R⌉ ; n_B = n_CB(n_CB+1)/2 ;
2  pour j de 1 à n_V faire
3      iter=1 ; m_1=1+(iter-1)m_R ; m_2=iter × m_R ; s(j)=x²_{m_1 j};
4      pour m de m_1 + 1 à m_2 faire
5          s(j)=s(j)+x²_{mj}
6      pour iter de 2 à n_RB faire
7          m_1=1+(iter-1)m_R ; m_2=iter × m_R;
8          pour m de m_1 à m_2 faire
9              s(j)=s(j)+x²_{mj}
10 pour κ de 1 à n_B faire
11     iter=1 ; m_1=1+(iter-1)m_R ; m_2=iter × m_R;
12     (K, L) = ψ_3(κ);
13     i_1=1+(K-1)m_C ; i_2=Km_C ; j_1=1+(L-1)m_C ; j_2=Lm_C;
14     si K >L alors
15         pour i de i_1 à i_2 faire
16             pour j de j_1 à j_2 faire
17                 k = j + (i-1)(i-2)/2 ; d(k)=s(j)+s(i);
18                 pour m de m_1 à m_2 faire
19                     d(k)=d(k)-2x_{mi} × x_{mj};
20     sinon
21         pour j de j_1 à j_2 faire
22             pour i de i_1 à j − 1 faire
23                 k = i + (j-1)(j-2)/2 ; d(k)=s(j)+s(i);
24                 pour m de m_1 à m_2 faire
25                     d(k)=d(k)-2x_{mi} × x_{mj};
26     pour iter de 2 à n_RB faire
27         m_1=1+(iter-1)m_R ; m_2=iter × m_R;
28         si K >L alors
29             pour i de i_1 à i_2 faire
30                 pour j de j_1 à j_2 faire
31                     k = j + (i-1)(i-2)/2;
32                     pour m de m_1 à m_2 faire
33                         d(k)=d(k)-2x_{mi} × x_{mj};
34         sinon
35             pour j de j_1 à j_2 faire
36                 pour i de i_1 à j − 1 faire
37                     k = i + (j-1)(j-2)/2;
38                     pour m de m_1 à m_2 faire
39                         d(k)=d(k)-2x_{mi} × x_{mj};
```

7.5.2.2 Parallélisation avec CUDA

Pour le calcul de la distance sur GPU, on utilise encore la formule donnée par le théorème 7.3.2. L'algorithme 13 est modifié : le calcul des normes se fait en calculant les vecteurs $\boldsymbol{s_A}$ et $\boldsymbol{s_B}$ pour chaque paire ($\boldsymbol{A}$, $\boldsymbol{B}$) de tuiles ($\boldsymbol{B}$ peut être la tuile $\boldsymbol{A}$). Si on veut calculer la norme pour tous les vecteurs d'abord, il faut copier la matrice $\boldsymbol{X}$ sur le GPU entièrement ce qui peut être lent pour de grandes matrices et allouer un vecteur $\boldsymbol{s}$ sur le GPU, ce qui peut être limitant pour des GPUs avec peu de mémoire. Afin de mieux superposer l'exécution des noyaux sur le GPU, on utilise deux *streams* et au lieu d'allouer une seule matrice $\boldsymbol{M}$ pour y stocker la distance au fur et à mesure du parcours des blocs de lignes $\mathcal{I}_i$, on alloue deux matrices $\boldsymbol{M_1}$ et $\boldsymbol{M_2}$. Pour une paire de tuile ($\boldsymbol{A}$, $\boldsymbol{B}$), le *stream* 1 copie la matrice $\boldsymbol{A}$ du CPU vers le GPU, calcule $\boldsymbol{s_A}$ puis calcule $\boldsymbol{M_1} = \boldsymbol{s_A}^\top \boldsymbol{1}^\top_{\boldsymbol{m_{CA}}}$. En même temps, le *stream* 2 copie la matrice $\boldsymbol{B}$ du CPU vers le GPU, calcule $\boldsymbol{s_B}$ puis calcule $\boldsymbol{M_2} = \boldsymbol{1}_{\boldsymbol{m_{CB}}} \boldsymbol{s_B}$. On copie toutes les lignes des matrices $\boldsymbol{A}$ et $\boldsymbol{B}$ afin de réduire le nombre de copies du CPU vers le GPU. Le calcul de $\boldsymbol{s_A}$ et $\boldsymbol{s_B}$ se fait alors sur l'ensemble des lignes afin d'utiliser au mieux le grand nombre de threads disponibles sur le GPU. Le calcul de $-2\boldsymbol{A}^\top \boldsymbol{B}$ sur les blocs de lignes est réparti entre les deux *streams* : le *stream* 1 est chargé des blocs de lignes d'indice impair $\mathcal{I}_{2i+1}$ et le *stream* 2 est chargé des blocs de lignes d'indice pair $\mathcal{I}_{2i}$. À la fin, les matrices $\boldsymbol{M_1}$ et $\boldsymbol{M_2}$ sont additionnées sur le GPU puis le *stream* 2 copie la matrice résultante sur le CPU dans une matrice $\boldsymbol{M_h}$. Pour le bloc $\boldsymbol{D}_{\kappa+1}$, pendant que le GPU calcule, le CPU va copier la matrice $\boldsymbol{M_h}$ qu'il a reçu à la fin du calcul de$\boldsymbol{D}_\kappa$ sur le GPU dans le vecteur de distances $\boldsymbol{D}$, ce qui permet de superposer le travail du CPU au travail sur GPU. Pour plus de clarté, la figure 7.24 illustre le calcul de la distance sur GPU en prenant une matrice $\boldsymbol{X}$ découpée en deux tuiles et en deux blocs de lignes :

$$\boldsymbol{X} = \begin{pmatrix} X_{11} & X_{12} \\ X_{21} & X_{22} \end{pmatrix}$$

7.5.2.3 Parallélisation avec MPI

L'algorithme 13 n'est pas adapté pour une architecture à mémoire distribuée. En effet, sur une telle architecture, l'échange d'informations entre processeurs n'est possible qu'en envoyant des messages par le réseau. Le calcul de la distance sur une architecture distribuée ne semble pas possible car il faut remplir une matrice triangulaire par bloc. Une adaptation directe de l'algorithme 10 conduit à une topologie impraticable avec un nombre variable de processeurs impliqués.

L'algorithme proposé ci-dessous est fondé sur des permutations circulaires. Une permutation circulaire d'un ensemble avec un décalage k à droite consiste à décaler les k derniers éléments en les plaçant au début de la liste et une permutation circulaire avec un décalage k à gauche consiste à placer les k premiers éléments à la fin de la liste. Par exemple, pour l'ensemble allant de 1 à 10, une permutation circulaire avec un décalage de trois à gauche donne 8, 9, 10, 1, 2, 3, 4, 5, 6 et 7 et à décalage de deux à droite donne 3, 4, 5, 6, 7, 8, 9, 10,1 et 2. Plus formellement, une permutation circulaire est une permutation σ des n entrées de l'ensemble $\{1, 2, \ldots, n\}$ telle que pour chaque entrée i :

$$\begin{aligned} \sigma(i) &= i + 1 \pmod n \quad \text{Permutation à droite} \\ \sigma(i) &= i - 1 \pmod n \quad \text{Permutation à gauche} \end{aligned} \tag{7.9}$$

En appliquant récursivement (7.9) k-fois, on obtient une permutation circulaire avec un décalage de k, noté σ_k et défini par :

$$\begin{aligned} \sigma_k(i) &= i + k \pmod n \quad \text{Permutation à droite} \\ \sigma_k(i) &= i - k \pmod n \quad \text{Permutation à gauche} \end{aligned} \tag{7.10}$$

FIGURE 7.24 – Version 3-Calcul de la distance sur GPU : illustration pour deux tuiles et deux blocs de lignes .

Considérons maintenant qu'on dispose de n_{P} processeurs. On découpe alors la matrice $\boldsymbol{X}$ en n_{P} tuiles $\boldsymbol{X}_{\cdot j}$ rectangulaires (autrement dit comportant toutes les lignes). On suppose que sur chaque processeur, on peut calculer efficacement la sous-matrice $\boldsymbol{D}_{ij} = d(\boldsymbol{X}_{\cdot i}, \boldsymbol{X}_{\cdot j})$, en adaptant l'algorithme 13.On commence l'algorithme en assignant au processeur i le calcul de la sous-matrice $\boldsymbol{D}_{ii}$. On réalise ensuite une permutation circulaire à droite pour le second indice de sorte qu'à la deuxième étape, le processeur i calcule $\boldsymbol{D}_{\sigma(i)i}$=$\boldsymbol{D}_{i+1i}$. En pratique, puisqu'il suffit de remplir la partie triangulaire inférieure de la matrice de distance, chaque processeur lorsqu'il a deux blocs de matrice d'indice $\cdot i$ et $\cdot j$ calcule la sous-matrice $\boldsymbol{D}_{i*j*}$ avec $i* = \max(i, j)$ et $j* = \min(i, j)$. La procédure circulaire continue, comme illustrée en table 7.2, jusqu'à ce que tous les blocs $\boldsymbol{D}_{ij}$ avec $j \leq i$ soient calculés (afin de ne calculer que la partie triangulaire inférieure de la matrice de distance). Le théorème 7.5.1 montre qu'on peut calculer la matrice triangulaire inférieure de distance en utilisant les permutations circulaires comme schéma de communication entre les processeurs.

Processeurs	1	2	...	i	...	n_{P}
Étape 1	$\boldsymbol{D}_{11}$	$\boldsymbol{D}_{22}$	...	$\boldsymbol{D}_{ii}$	...	$\boldsymbol{D}_{n_{\mathrm{P}}n_{\mathrm{P}}}$
Étape 2	$\boldsymbol{D}_{21}$	$\boldsymbol{D}_{32}$	...	$\boldsymbol{D}_{i+1i}$	...	$\boldsymbol{D}_{n_{\mathrm{P}}1}$
Étape 3	$\boldsymbol{D}_{31}$	$\boldsymbol{D}_{42}$	...	$\boldsymbol{D}_{i+2i}$	...	$\boldsymbol{D}_{n_{\mathrm{P}}2}$
...						
Étape k	$\boldsymbol{D}_{\sigma_{k-1}(1)1}$	$\boldsymbol{D}_{\sigma_{k-1}(2)2}$	...	$\boldsymbol{D}_{\sigma_{k-1}(i)i}$	...	$\boldsymbol{D}_{n_{\mathrm{P}}\sigma_{k-1}(n_{\mathrm{P}})}$

TABLE 7.2 – Procédure de permutation circulaire successive sur les blocs .

Théorème 7.5.1. *La procédure de permutations cycliques décrite en table 7.2 s'arrête après* $\frac{(n_{\mathrm{P}}+1)}{2}$ *étapes.*

Démonstration. Lorsqu'on découpe la matrice $\boldsymbol{X}$ en n_{P} tuiles, la partie triangulaire inférieure de la matrice de distance contient $\frac{n_{\mathrm{P}}(n_{\mathrm{P}}+1)}{2}$ bloc car il y a un bloc par paire de tuile $(\boldsymbol{X}_{\cdot i},\boldsymbol{X}_{\cdot j})$. Or à chaque étape, la procédure de permutation circulaire calcule n_{P} blocs de distance. Il faut donc au moins $\lceil (n_{\mathrm{P}}+1)/2 \rceil$étapes. Montrons maintenant que les blocs calculés jusqu'à l'étape $\lceil (n_{\mathrm{P}}+1)/2 \rceil$sont différents. À l'étape k, le processeur i calcule $\boldsymbol{D}_{\sigma_{k-1}(i)i}$ (σ_0 étant la fonction identité). Or pour un ensemble à n éléments, $\sigma_{k-1}(i) = i$ lorsque $k-1 = n$. Le processeur i recalcule donc les distances à partir de l'étape $n_{\mathrm{P}}+1$. Un processeur ne recalcule donc pas un même bloc. Reste à montrer qu'un processeur i ne calcule pas un bloc $\boldsymbol{D}_{\sigma_{k-1}(i)i}$ qui équivaut à un bloc $\boldsymbol{D}_{\sigma_{k'-1}(j)j}$ calculé sur un processus j. Comme $i \neq j$, ce cas ne se produit que si $i = \sigma_{k'-1}(j)$ et $j = \sigma_{k-1}(j)$. Le système présenté en (7.11) traite le cas où $k = k'$ (autrement dit, on calcule des blocs identiques durant la même étape) aboutit au cas n_{P} divisible par $2(k-1)$ donc pair. En effet, si on prend $k = \lceil (n_{\mathrm{P}}+1)/2 \rceil$,on calcule plusieurs fois le même bloc sur des processeurs différents. Le système donné en 7.12 présente le cas où on prend $k \neq k'$ et aboutit à la conclusion que $k + k' - 2$ divise n_{P}. Or, si on considère qu'il n'y a que $\lceil (n_{\mathrm{P}}+1)/2 \rceil$étapes, on a alors $k \leq \lceil (n_{\mathrm{P}}+1)/2 \rceil$et $k' \leq \lceil (n_{\mathrm{P}}+1)/2 \rceil$donc $k + k' - 2 \leq \lceil (n_{\mathrm{P}}+1)/2 \rceil$.Dans le cas $n_{\mathrm{P}} = 2p$ pair, on obtient $k+k'-2 \leq 2p(= n_{\mathrm{P}})$ et on retrouve le cas précédent. Dans le cas $n_{\mathrm{P}} = 2p+1$ impair, on obtient $k + k' - 2 \leq 2p < n_{\mathrm{P}}$, ce qui est en contradiction avec $k + k' - 2$ diviseur de n_{P}.

$$\begin{cases} i = \sigma_{k-1}(j) = j + k - 1 \pmod n \\ j = \sigma_{k-1}(i) = i + k - 1 \pmod n \end{cases} \Leftrightarrow \begin{cases} i = j + k - 1 \pmod n \\ j = j + 2k - 2 \pmod n \end{cases} \Leftrightarrow \begin{cases} i = j + k - 1 \pmod n \\ 2(k-1) = 0 \pmod n \end{cases} \tag{7.11}$$

$$\begin{cases} i = \sigma_{k'-1}(j) = j + k' - 1 \pmod n \\ j = \sigma_{k-1}(i) = i + k - 1 \pmod n \end{cases} \Leftrightarrow \begin{cases} i = j + k' - 1 \pmod n \\ j = j + k + k' - 2 \pmod n \end{cases} \Leftrightarrow \begin{cases} i = j + k' - 1 \pmod n \\ k + k' - 2 = 0 \pmod n \end{cases} \tag{7.12}$$

Donc $\lceil (n_{\mathrm{P}}+1)/2 \rceil$ étapes sont suffisantes pour calculer tous les blocs. □

Le théorème 7.5.1 a deux conséquences :

— Si n_{P} est pair, à la dernière itération seuls $\lceil (n_{\mathrm{P}}+1)/2 \rceil$ blocs ont besoin d'être calculés, comme montré ci-dessous dans le cas n_{P}=4 (en italique sont présentés les blocs en double qui ne doivent pas être calculés) :

(1,1) (2,2) (3,3) (4,4)
(1,2) (2,3) (3,4) (4,1)
(1,3) (2,4) *(3,1)* *(4,2)*

— Si n_{P} est impair, à chaque étape n_{P} blocs doivent être calculés, comme illustré ci-dessous dans le cas n_{P}=5 :

(1,1) (2,2) (3,3) (4,4) (5,5)
(1,2) (2,3) (3,4) (4,5) (5,1)
(1,3) (2,4) (3,5) (4,1) (5,2)

En pratique, les permutations cycliques sont réalisées par des procédures d'envoi et de réception dans une topologie cartésienne 1-D périodique dans laquelle le processeur i a deux voisins : le processeur $i-1 \pmod{n_{\mathrm{P}}}$ et le processeur $i+1 \pmod{n_{\mathrm{P}}}$. À une étape k, le processeur i calcule $\boldsymbol{D}_{\sigma_{k-1}(i)i}$: il a donc besoin de $\boldsymbol{X}_{\cdot i}$ qu'il a reçu au commencement de l'algorithme et de $\boldsymbol{X}_{\cdot\sigma_{k-1}(i)}$ qu'il n'a pas et qui doit donc lui être envoyé. Reprenons le processus à partir de la deuxième étape : le processeur i calcule $\boldsymbol{D}_{i+1i}$ (respectivement $\boldsymbol{D}_{i1}$ si $i = n_{\mathrm{P}}$). Or le processeur $i+1$ (respectivement le processus 1 si $i = n_{\mathrm{P}}$) a la tuile $\boldsymbol{X}_{i+1}$ (respectivement $\boldsymbol{X}_1$) : il peut donc l'envoyer au processeur i. De même, le processeur i envoie $\boldsymbol{X}_{\cdot i}$ au processeur $i-1$ (ou processeur n_{P} si $i = 1$). À la troisième étape, le processeur i calcule $\boldsymbol{D}_{i\sigma_2(i)}$, or $\sigma_2(i) = \sigma_1(i+1)$ car ce sont des permutations circulaires dans le même sens. Cela veut dire que le processeur $i+1$ dispose de $\boldsymbol{X}_{\cdot\sigma_1(i+1)} = \boldsymbol{X}_{\cdot\sigma_2(i)}$ donc il peut l'envoyer. De manière similaire, la tuile $\boldsymbol{X}_{\cdot\sigma_1 i}$ que le processeur i a reçue à la seconde étape est demandée par le processeur $i-1 \pmod{n}$. À chaque étape $k > 1$, le processeur i reçoit donc du processeur $i+1 \pmod{n_{\mathrm{P}}}$ la tuile $\boldsymbol{X}_{\cdot\sigma_{k-1}(i)}$, il calcule la distance puis il transmet au processeur $i-1 \pmod{n_{\mathrm{P}}}$ la tuile $\boldsymbol{X}_{\cdot\sigma_{k-1}(i)}$.

L'algorithme distribué pour le calcul de la distance euclidienne est donné à l'algorithme 14. Les permutations circulaires sont itérées dans la boucle commençant à la ligne 4. La boucle commençant en ligne 5 sert à indiquer que les trois opérations internes sont exécutées par chaque processeur indépendamment. Si ce n'est pas la première étape, chaque processeur commence par recevoir la tuile nécessaire au calcul de la distance en provenance de son voisin de droite à la ligne 7. Puis, chaque processeur transmet cette tuile à son voisin de gauche à la ligne 8 et enfin, il calcule la distance à la ligne 9. La réception est placée avant le calcul car elle est nécessaire pour le calcul de la distance. L'intérêt de commencer l'envoi avant le calcul est que si on utilise un envoi asynchrone non bloquant, on peut recouvrir l'envoi par le calcul de la distance.

Algorithme 14 : Algorithme distribué du calcul de la distance euclidienne.

Entrée : $\boldsymbol{X}$ matrice de taille $n_{\mathrm{T}} \times n_{\mathrm{V}}$;
n_{P} le nombre de processeurs organisés par une topologie cartésienne 1-D périodique
Sortie : $\boldsymbol{D}$ vecteur de taille $\frac{n_{\mathrm{V}}(n_{\mathrm{V}}-1)}{2}$

1 Partitionner $\boldsymbol{X}$ en n_{P} tuiles $(\boldsymbol{X}_{\cdot i})_{i\in\{1,\ldots,n_{\mathrm{P}}\}}$;
2 Assigner au processeur I la tuile $\boldsymbol{X}_{\cdot I}$ et calculer la sous-matrice $\boldsymbol{D}_{II}$;
3 Créer une copie locale de $\boldsymbol{X}_{\cdot i}$ pour chaque processeur i;
4 **pour** *K de 1 à* $\left\lceil \frac{(n_{\mathrm{P}}+1)}{2} \right\rceil$ **faire**
5 **pour** *Pour chaque processeur* **faire**
6 **si** *k>1* **alors**
7 Recevoir du processeur $I+1 \pmod{n_{\mathrm{P}}}$ la tuile $\boldsymbol{X}_{\cdot\sigma_{K-1}(I)}$;
8 Envoyer au processeur $I-1 \pmod{n_{\mathrm{P}}}$ la tuile $\boldsymbol{X}_{\cdot\sigma_{K-1}(I)}$;
9 Calculer $\boldsymbol{D}_{I\sigma_{K-1}(I)}$;

Dans ce chapitre, nous avons optimisé le calcul de la matrice de la distance par un découpage en tuiles. Nous avons proposé plusieurs parallélisations possibles adaptées à des architectures à mémoire partagée, à des architectures avec des GPUs NVIDIA et à des architectures à mémoire distribuée. La dernière version de l'algorithme est un algorithme *cache-aware* pour lequel nous proposons un test pour déterminer la taille de tuile sur une architecture donnée. Les résultats numériques sont décrits en section 8.2.

Troisième partie

Résulats

Chapitre 8

Résultats numériques

Dans ce chapitre, nous comparons les différents algorithmes de calcul de la distance en séquentiel et en parallèle. Puis, sont présentées les accélérations et performances obtenues pour les algorithmes de parallélisation de la distance. Enfin, sont montrés les résultats obtenus pour l'algorithme de Ward complet. Tous les algorithmes ont été implémentés en FORTRAN, excepté le calcul de la distance sur GPU. La parallélisation avec CUDA a été écrite en C et la fonction C est appelée dans le code FORTRAN. Nous avons choisi le langage FORTRAN pour bénéficier de ses fonctions vectorielles internes. Pour les tests avec OpenMP, chaque *thread* a été attaché à un CPU (avec OMP_PROC_BIND=1).

8.1 Caractéristiques des machines utilisées

Les tests ont été réalisés sur trois environnements de travail définis ci-dessous :

COMP1 un DELL PRECISION avec 16 cœurs Intel(R) Xeon(R) E5-1660 avec une fréquence d'horloge à 3,20 GHz et un GPU NVIDIA Quadro M2000 avec 768 cœurs CUDA et une fréquence d'horloge à 1,16 GHz. La figure 8.1 montre la topographie de la mémoire pour cet ordinateur : chaque cœur a deux CPUs se partageant 256 Ko de cache L2 et les huit cœurs se partagent un cache L3 de 20 Mo. La mémoire totale est de 64 Go pour le CPU et 4 Go pour le GPU.

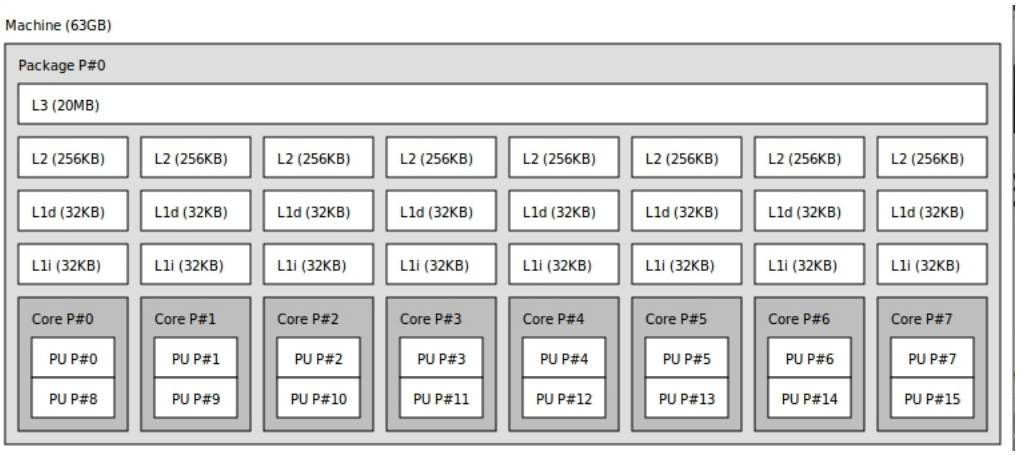


FIGURE 8.1 – Organisation de la mémoire de COMP1.

COMP2 un DELL PowerEdge R930 muni de processeurs Intel Xeon E7-8890 avec une fréquence d'horloge à 2,50 GHz. L'ordinateur est constitué de quatre sockets de 18 cœurs chacune (donc 72 cœurs au total). Chaque socket a un cache L_3 privé de 45 Mo partagé par tous

les cœurs sur la socket et chaque cœur a un cache L_2 privé de 256 Ko et un cache L_1 de 32 Ko. La RAM totale est de 3 To.

COMP3 un DELL PRECISION RACK7910 muni de 10 cœurs Intel Xeon E5-2640 avec une fréquence d'horloge de 2,40 GHz et un GPU NVIDIA Quadro P5000 avec 2 560 cœurs CUDA. La mémoire RAM totale est de 62 Go pour le CPU et 16 Go pour le GPU.

8.2 Calcul de la distance euclidienne

8.2.1 Comparaison des algorithmes séquentiels

Le nombre de lignes a été fixé à $n_{\mathrm{T}} = 1\,000$. Le nombre de colonnes n_{V} varie entre 10 000 et 100 000. On compare les algorithmes suivants :

ALGA l'algorithme 7 qui calcule la distance avec une double boucle,

ALGB l'algorithme 8 qui vectorise le calcul de la distance, implémenté à l'aide de la fonction SPREAD pour calculer $\boldsymbol{A}$,

ALGC l'algorithme 8 calculant la distance par des opérations vectoriels en utilisant la fonction MATMUL pour calculer $\boldsymbol{A}$,

ALGD l'algorithme 9 calculant la distance par des opérations matricielles utilisant la fonction MATMUL pour calculer $\boldsymbol{X}\boldsymbol{X}^{\top}$,

ALGE l'algorithme 9 calculant la distance par des opérations matricielles en utilisant la fonction DGEMM de la librairie BLAS pour calculer $\boldsymbol{X}\boldsymbol{X}^{\top}$,

ALGF l'algorithme 10 qui opère un découpage en tuiles rectangulaires et utilisant la fonction SUM pour calculer la distance entre les paires de colonnes des tuiles,

ALGG l'algorithme 10 avec découpage en tuiles rectangulaires et utilisant la fonction DOT_PRODUCT pour calculer la distance entre deux colonnes,

ALGH l'algorithme 11 qui correspond au découpage de la matrice $\boldsymbol{X}$ en tuiles rectangulaires utilisant la fonction MATMUL pour le calcul matriciel,

ALGI l'algorithme 11 qui découpe la matrice en tuiles rectangulaires et calcule la distance entre els tuiles avec la fonction DGEMM de la librairie BLAS pour le calcul matriciel.

	Temps CPU		
n_{V}	10 000	50 000	90 000
ALGA	75, 14	1 901,62	6 386,09
ALGB	820,63	22 387,58	*OOM*
ALGC	497,88	12 028,87	*OOM*
ALGD	145,87	3 777,91	*OOM*
ALGE	134,52	3 366,31	12 254,55
ALGF	*42,82*	*1 107,72*	*3 834,36*
ALGG	59, 13	1 517,89	5 582,21
ALGH	44,36	1 128,54	3 977,52
ALGI	45,61	1 127, 19	3 934,21

TABLE 8.1 – Comparaison des temps CPU (en secondes) des différents algorithmes de calcul de distance sur COMP1.(*OOM* dépassement mémoire)

Le tableau 8.1 donne les temps de calcul en secondes pour les différents algorithmes envisagés en prenant n_{T}=1 000. Pour les algorithmes vectoriels (ALGB et ALGC), on constate que les implémentations ne permettent pas de calculer pour un grand nombre de colonnes à cause de la limitation mémoire. De plus, comme on copie de grosses matrices la fonction SPREAD est très lente. La version matricielle (ALGD et ALGE) est plus lente que la version simple avec la double boucle (ALGA). Le découpage en tuiles est en revanche plus efficace (ALGF, ALGG, ALGH

et ALGI) : les temps de calcul ont diminué par rapport à ALGA. Cependant, l'utilisation du produit scalaire (ALGG) ralentit l'exécution. L'utilisation de la fonction *sum* (ALGF) semble plus efficace que l'utilisation d'opérations matricielles (ALGH et ALGI). La figure 8.2 compare les performances en GFlops des différents algorithmes en prenant le nombre d'opérations égal à $n_T n_V^2$ pour les algorithmes vectoriels (ALGB et ALGC) et matriciels (ALGD et ALGE) et le nombre d'opérations égale à $n_T n_V^2/2$ pour les autres algorithmes qui exploitent la symétrie de la matrice de distance. En concordance avec la table 8.1, la figure confirme que ALGF c'est-à-dire par découpage en tuile avec la fonction SUM donne les meilleures performances en séquentiel avec un maximum de 1,2 GFlops, suivi par ALGH puis ALGI qui ont des performances très proches.

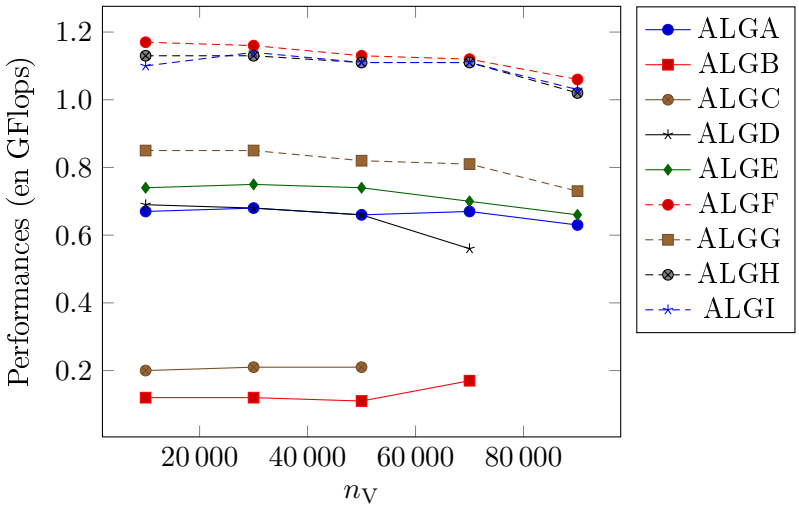


FIGURE 8.2 – Comparaison des algorithmes séquentiels de calcul de la distance sur COMP1.

Comme les temps d'exécution des algorithmes ALGF, ALGG, ALGH et ALGI dépendent de la taille de la tuile choisie, nous avons fait varier cette taille de tuile de 128 à 1 024, en choisissant des multiples de 32 (afin d'être en accord avec la taille des *warps* sur GPU). La figure 8.3(a) compare les performances de ALGF pour différents tailles de problèmes, nous observons que pour $n_V \leq 50\,000$ les meilleures performances atteignent environ 1,20 GFlops pour $m_C{}^*$=512. Pour n_V=70 000, $m_C{}^*$=1 024 donne une performance de 1,19 GFlops. Enfin, pour n_V=90 000, la meilleure performance est de 1,06 GFlops pour $m_C{}^*$=128. La figure 8.3(b) montre l'influence de la taille de la tuile sur les performances de ALGG pour différentes tailles : nous constatons que pour $n_V \leq 50\,000$, la performance optimale est autour de 0,85 GFlops pour $m_C{}^*$=512, moindre que la performance obtenue pour ALGF. Pour $n_V > 50\,000$, $m_C{}^*$=1 024 donne les meilleures performances (0,87 GFlops pour n_V=70 000 et 0,79 GFlops pour n_V=90 000). Nous observons ainsi que ALGG est moins performant que ALGF.

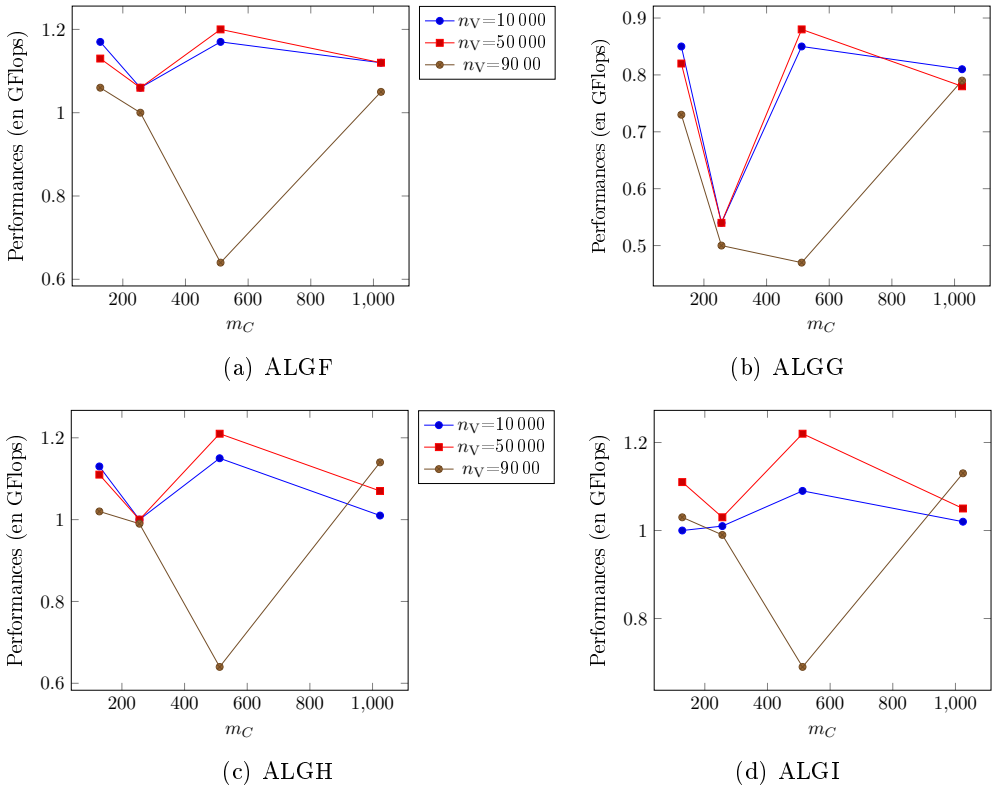
La figure 8.3(c) compare les performances de ALGH pour différentes tailles de tuiles. Nous constatons que pour $n_V \leq 50\,000$ la performance optimale est entre 1,15 GFlops et 1,20 GFlops et est atteinte pour $m_C{}^*$=512. En revanche, pour $n_V > 50\,000$, la performance est au-dessus de 1,15 GFlops et est atteinte pour $m_C{}^*$=1 024. Les performances maximales de ALGH sont un peu en dessous des performances de ALGF. La figure 8.3(d) présente l'influence de la taille de la tuile sur les performances de ALGI. Nous obtenons pour $n_V = 10\,000$ une performance maximale de 1,1 GFlops pour $m_C{}^*$=128. Pour $n_V > 50\,000$, la performance maximale est atteinte pour $m_C = 1\,024^*$ et est aux alentours 1,15 GFlops. Pour les problèmes de taille moyenne, la performance optimale au-dessus de 1,17 GFlops s'obtient avec $m_C{}^*$=512. Nous constatons que ALGH et ALGI ont des performances très similaires, légèrement inférieures aux performances de ALGG.

La taille des tuiles $m_C{}^*$ donnant les meilleures performances est donnée pour chacun des

quatre algorithmes à la table 8.2 en fonction de la taille du problème (le nombre de lignes ayant été fixé à n_T=1 000).

	$m_C{}^*$		
n_T	10 000	50 000	100 000
ALGF	512	512	128
ALGG	512	512	1 024
ALGH	512	512	1 024
ALGI	128	521	1 024

TABLE 8.2 – $m_C{}^*$ pour ALGF, ALGG, AGLH et ALGI



(a) ALGF (b) ALGG

(c) ALGH (d) ALGI

FIGURE 8.3 – Effet de la taille de la tuile sur la performance en GFlops sur COMP1.

8.2.2 Comparaison des algorithmes parallélisés

Cette section compare les différentes parallélisations du calcul de la distance :

ALG_SHARE la troisième version de la parallélisation avec OpenMP présentée en section 7.5.2.1 avec une implémentation matricielle c'est-à-dire avec découpage de la matrice $\boldsymbol{X}$ en blocs de lignes et blocs de colonnes et parcours des tuiles de la matrice de distance,

ALG_GPU la première version de la parallélisation avec CUDA présentée en section 7.3.2 avec découpage en blocs de colonnes et parcours des tuiles de la matrice $\boldsymbol{X}$ par une double boucle,

ALG_DIS la troisième version de la parallélisation avec MPI présentée en section 7.5.2.3 correspondant à un découpage des lignes et des colonnes en blocs et parcours des tuiles de la matrice $\boldsymbol{D}$,

ALG_HYBR la parallélisation avec CUDA+OpenMP présentée en section 7.4.3 correspondant à un découpage tuiles de colonnes avec superposition des noyaux CUDA grâce à OpenMP,

ALG1 la première version de la parallélisation avec OpenMP présentée en section 7.3.1 qui correspond à un découpage en tuiles de colonnes et parcours des tuiles par une double boucle,

ALG2 la deuxième version de la parallélisation avec OpenMP présentée en section 7.4.1 où la matrice $\boldsymbol{X}$ est découpé en tuiles de colonnes mais les tuiles sont parcourues avec une seule boucle,

ALG3 la troisième version de la parallélisation avec OpenMP présentée en section 7.5.2.1 qui correspond à un découpage de $\boldsymbol{X}$ en blocs de lignes et de colonnes avec une implémentation vectorielle du calcul de la distance entre les blocs et un parcours des tuiles avec une seule boucle,

ALG4 la deuxième version de la parallélisation avec CUDA présentée en section 7.4.2 qui découpe la matrice $\boldsymbol{X}$ en blocs de colonnes et parcourt les tuiles de la matrice $\boldsymbol{D}$ avec une seule boucle avec déroulement de cette boucle pour lancer plusieurs noyaux CUDA en même temps,

ALG5 la troisième version de la parallélisation avec CUDA présentée en section 7.5.2.2 qui correspond à découpage en blocs de lignes et de colonnes de la matrice $\boldsymbol{X}$.

Le nombre de lignes vaut soit 1 000 pour simuler la parcellisation individuelle soit 10 000 pour simuler le cas de parcellisations de groupe. Le nombre de colonnes a été pris entre 10 000 et 120 000. Les tests avec OpenMP ont été réalisés en fixant un *thread* par cœur. La taille des tuiles et des blocs de ligne est donnée pour chaque algorithme à la table 8.3. Pour ALG5, la profondeur du déroulement de la boucle (c'est-à-dire le nombre de noyaux CUDA lancés par itération de la boucle) a été fixée à cinq.

La table 8.4 compare les algorithmes parallélisés avec OpenMP c'est-à-dire les algorithmes ALG_SHARE, ALG1, ALG2 et ALG3 en fonction du temps d'exécution en secondes sur COMP1 en double précision. Pour ALG1 quelle que soit la taille du problème, il y a une stagnation du temps d'exécution à partir de huit *threads*, ce qui était attendu puisque sans linéarisation des indices matriciels, il y a un déséquilibre des charges entre les *threads*. Pour ALG3, passer de huit *threads* à seize *threads* diminue peu le temps de calcul. Pour ALG2 et ALG3, il n'y a pas de stagnation entre huit et seize *threads*, ce qui montre que la linéarisation des indices matriciels permet une meilleure répartition du travail entre les *threads*. En revanche, ALG2 est plus lent que ALG1 excepté pour 16 *threads*. Pour n_{T}=10 000, ALG3 est plus rapide que ALG2 excepté pour n_{V}=100 000, ce qui montre que le découpage en bloc de lignes est efficace. Pour n_{T}=100 000, les temps très longs proviennent certainement des paramètres (m_R, m_C) inadaptés car ils ont été choisi en fonction d'un plus petit problème. Le phénomène n'est pas reproduit pour n_{T}=1 000 alors qu'il y a un découpage en blocs de lignes et est attribuable à l'utilisation des opérations vectorielles plus lentes que les opérations matricielles comme constaté en section

	Nombre de threads	m_R	m_C
ALG_SHARE	1 à 8	64	256
ALG_SHARE	16	128	512
ALG1	1 à 16	N.R.	1 024
ALG2	1 à 16	N.R.	1 024
ALG3	1 à 4	256	1 024
ALG3	8	64	256
ALG3	16	128	512
ALG_GPU	N.R.	N.R.	2 048
ALG4	N.R.	N.R	1 024
ALG5	N.R	256	1 024

(a) Double précision

	Nombre de threads	n_{T}	m_R	m_C
ALG_SHARE	1 à 8	-	64	256
ALG_SHARE	16	-	128	1 024
ALG1	1 à 16	-	N.R.	1 024
ALG2	1 à 16-		N.R.	1 024
ALG3	1	-	1 000	2 048
ALG3	2	-	1 000	512
ALG3	4 à 16	-	1 000	256
ALG_GPU	N.R.	1 000	N.R.	1 024
ALG_GPU	N.R.	10 000	N.R.	2 048
ALG4	N.R.	-	N.R	1 024
ALG5	N.R	-	128	1 024

(b) Simple précision

TABLE 8.3 – Tailles des tuiles et des blocs de lignes choisies pour les différents algorithmes sur COMP1
$-n_{\mathrm{T}} \in \{1\,000, 10\,000\}$; N.R. : non requis.

8.2.1. L'algorithme le plus rapide quel que soit le nombre de *threads* employés est ALG_SHARE qui correspond à un découpage en lignes et en colonnes avec des opérations matricielles.

La table 8.4 compare les algorithmes ALG_SHARE, ALG1, ALG2 et ALG3 en fonction du temps d'exécution en secondes sur COMP1 en simple précision. Pour ALG3, nous constatons une stagnation du temps de calcul entre huit et seize *threads* malgré la linéarisation. Pour ALG1, il n'y a pas de stagnation mais seulement une faible amélioration du temps de calcul entre huit et seize *threads*. On retrouve toutefois que ALG_SHARE est plus rapide que les trois autres algorithmes quel que soit le nombre de *threads* employés et la taille du problème comme le montre la figure 8.4.

La table 8.6 compare les parallélisations avec CUDA c'est-à-dire les algorithmes ALG_GPU, ALG4 et ALG5 en fonction de leur temps d'exécution sur COMP1 en simple et double précision. En comparant ALG4 et ALG5, nous constatons que ALG5 est plus lent que ALG4 malgré un découpage en blocs de lignes et le portage sur GPU du calcul de la norme des colonnes. Toutefois, les temps d'exécution de ALG4 sont montrés avec un déroulement de boucle permettant de superposer plus de noyaux CUDA sur le GPU que ALG5. Contrairement à la parallélisation avec OpenMP, l'algorithme le plus performant est ALG_GPU qui ne fait intervenir ni linéarisation des blocs de la matrice de distance ni le découpage en blocs de lignes. Ceci s'explique par le fait que le parcours de la double boucle permet de réduire le nombre de copies du CPU vers le GPU puisque la tuile $\boldsymbol{A}$ n'est copiée qu'une seule fois pour toutes les paires de tuiles ($\boldsymbol{A}$,$\boldsymbol{B}$).

Ainsi, nous avons établi que l'algorithme le plus rapide en séquentiel est ALG_-

	n_T=1 000				n_T=10 000			
	ALG_SHARE	ALG1	ALG2	ALG3	ALG_SHARE	ALG1	ALG2	ALG3
Séquentiel	35,39	42,27	56,92	77,12	346,09	448,96	785,47	761,29
2 threads	20,66	25,21	37,16	39,30	195,40	277,85	578,81	386,19
4 threads	11,01	16,30	18,86	19,90	106,94	165,13	285,27	201,58
8 threads	5,50	11,08	10,42	10,62	53,41	113,63	152,26	105,73
16 threads	3,24	9,96	6,21	10,30	30,80	125,00	82,73	102,42

(a) n_V=10 000

	n_T=1 000				n_T=10 000			
	ALG_SHARE	ALG1	ALG2	ALG3	ALG_SHARE	ALG1	ALG2	ALG3
Séquentiel	860,28	1 049,69	1 423,14	1 905,96	8 275,41	11 218,68	19 449,94	19 050,43
2 threads	498,70	535,41	916,08	967,39	4 789,56	6 029,73	14 413,29	9 510,96
4 threads	273,60	322,38	461,66	492,90	2 618,34	3 256,97	6 983,61	5 011,20
8 threads	138,45	196,99	245,98	261,98	1 312,85	2 008,34	3 571,43	2 610,63
16 threads	79,72	193,25	125,27	251,15	761,68	2 005,91	1 885,21	2 528,98

(b) n_V=50 000

	n_T=1 000				n_T=10 000			
	ALG_SHARE	ALG1	ALG2	ALG3	ALG_SHARE	ALG1	ALG2	ALG3
Séquentiel	3 461,98	4 199,21	5 689,40	7 447,80	28 046,13	44 918,72	77 929,03	75 309,11
2 threads	2 009,98	2 169,33	3 653,47	3 772,01	20 021,06	24 476,43	56 465,27	38 185,36
4 threads	1 104,40	1 266,88	1 846,89	1 967,33	10 548,03	12 681,65	27 850,51	20 027,44
8 threads	563,93	743,79	985,51	1 047,34	5 292,04	7 478,98	14 211,84	10 406,39
16 threads	326,52	756,15	498,28	997,59	3 082,66	7 916,87	7 564,88	10 153,14

(c) n_V=100 000

TABLE 8.4 – Comparaison des temps CPU (en secondes) des algorithmes parallélisés avec OpenMP en double précision sur COMP1 .

	n_T=1 000				n_T=10 000			
	ALG_SHARE	ALG1	ALG2	ALG3	ALG_SHARE	ALG1	ALG2	ALG3
Séquentiel	31,49	42,33	56,29	70,11	308,66	429,78	640,21	717,06
2 threads	16,24	25,75	25,43	35,48	158,64	254,95	301,00	368,96
4 threads	8,51	15,38	13,37	18,55	82,94	156,09	159,61	187,66
8 threads	4,26	10,16	7,02	9,29	41,59	102,37	85,43	94,30
16 threads	3,03	11,27	4,49	9,37	29,35	89,81	50,89	94,06

(a) n_V=10 000

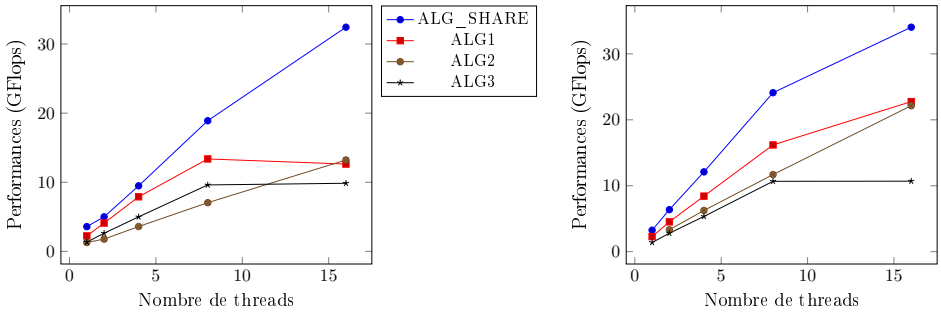
	n_T=1 000				n_T=10 000			
	ALG_SHARE	ALG1	ALG2	ALG3	ALG_SHARE	ALG1	ALG2	ALG3
Séquentiel	788,48	1047,50	1 408,84	1 753,25	7 685,21	11 049,00	15 802,90	17 844,58
2 threads	407,14	532,40	637,60	889,20	3 925,91	5 745,86	7 558,45	8 997,49
4 threads	213,16	295,53	332,77	463,29	2 068,17	3 053,89	3 998,86	4 685,00
8 threads	107,58	165,41	167,90	231,85	1 033,14	1 685,41	2 127,51	2 344,90
16 threads	76,80	123,70	98,67	232,61	731,46	1 305,56	1 130,62	2 338,68

(b) n_V=50 000

	n_T=1 000				n_T=10 000			
	ALG_SHARE	ALG1	ALG2	ALG3	ALG_SHARE	ALG1	ALG2	ALG3
Séquentiel	3 190,54	4 189,79	5 639,69	7 015,09	30 769,54	43 897,01		71 993,25
2 threads	1 627,61	2 161,40	2 559,19	3 547,10	15 703,23	22 142,14	30 124,35	35 897,72
4 threads	863,56	1 155,95	1 347,10	1 852,93	8 267,79	11 872,82	16 002,26	18 739,04
8 threads	442,38	619,25	686,38	926,80	4 147,31	6 177,16	8 551,53	9 373,73
16 threads	319,32	446,07	405,22	931,03	2 937,05	4 392,98	4 516,54	9 354,35

(c) n_V=100 000

TABLE 8.5 – Comparaison des temps CPU (en secondes) des algorithmes parallélisés avec OpenMP en simple précision sur COMP1.



(a) Double précision (b) Simple précision

FIGURE 8.4 – Comparaison des performances (en GFlops) des algorithmes parallélisés avec OpenMP en double précision pour un problème de taille 10 000 × 100 000

n_T	n_V	ALG_GPU	ALG4	ALG5
1000	10000	2,93	3,63	3,43
	50000	62,49	68,85	78,56
	100000	247,60	272,96	314,83
10000	10000	23,79	26,40	29,96
	50000	533,79	597,63	696,01
	100000	2115,31	2368,37	2760,81

(a) Double précision

n_T	n_V	ALG_GPU	ALG4	ALG5
1 000	10 000	0,84	1,28	1,92
	50 000	13,48	28,19	48,17
	100 000	53,19	115,23	195,53
10 000	10 000	2,59	8,33	8,90
	50 000	52,58	187,26	206,10
	100 000	205,03	728,92	786,95

(b) Simple précision

TABLE 8.6 – Comparaison des temps CPU (en secondes) des algorithmes parallélisés avec CUDA sur COMP1.

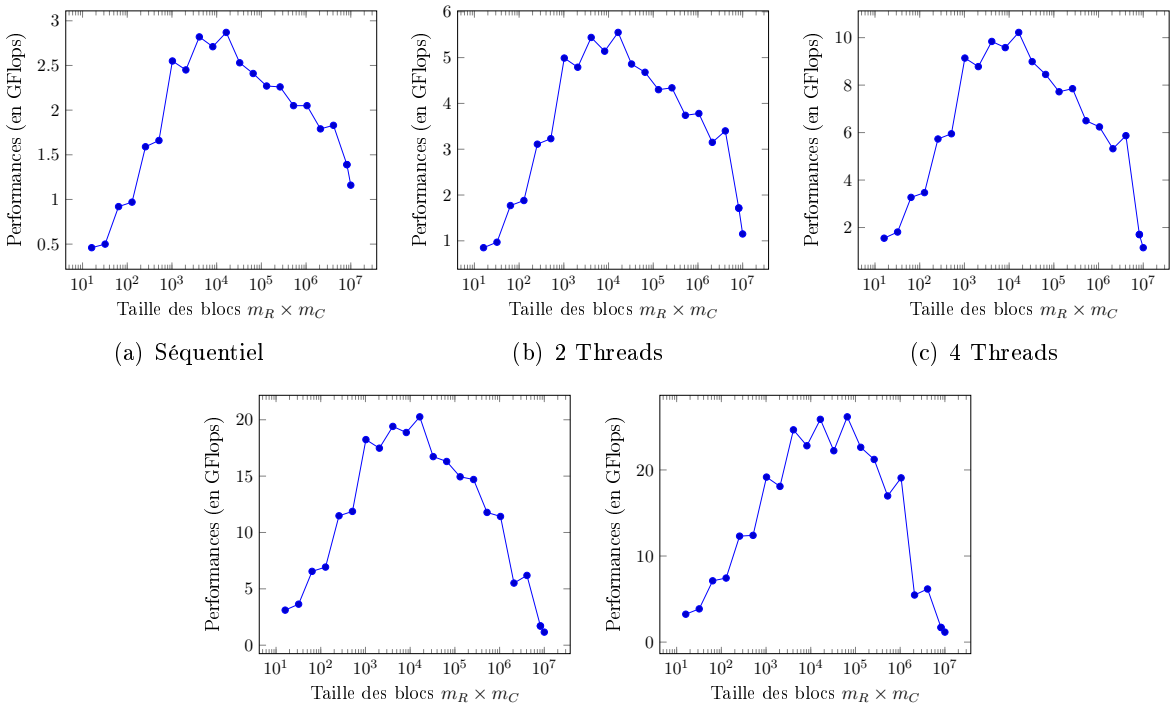
SHARE et donc son temps d'exécution servira de référence pour le calcul de l'accélération. ALG_SHARE permet également la meilleure parallélisation avec OpenMP. Pour la parallélisation sur GPU, on gardera dans la suite des résultats ALG_GPU.

8.2.3 Choix de la taille de la tuile et des blocs

Les performances des algorithmes sélectionnés à savoir ALG_SHARE, ALG_GPU, ALG_HYBR et ALG_DIST dépendent du choix de la taille des blocs de lignes et de colonnes choisis. Nous avons donc développé un banc de tests pour déterminer $(m_R{}^*, m_C{}^*)$ donnant les meilleures performances.

La figure 8.5 montre l'influence de la taille des tuiles et des blocs sur les performances de ALG_SHARE en double précision sur COMP1 pour un problème de taille n_T=1 000 et n_V=10 000. Les performances augmentent lorsque lorsque m_R et m_C augmentent jusqu'à atteindre leur maximum pour $m_R{}^* = 64$ et $m_C{}^* = 256$ en séquentiel et en parallèle jusqu'à huit *threads*. Pour seize *threads*, la performance optimale est atteinte pour $m_R{}^* = 128$ et $m_C{}^* = 512$. Avec seize *threads*, toute la socket est occupée ce qui explique la différence entre les performances. La figure 8.6 montre l'évolution des performances de ALG_SHARE en simple précision pour un problème de taille n_T=1 000 et n_V=10 000. Nous trouvons que les performances optimales sont obtenues quel que soit le nombre de *threads* pour $m_R{}^* = 64$ et $m_C{}^* = 256$.

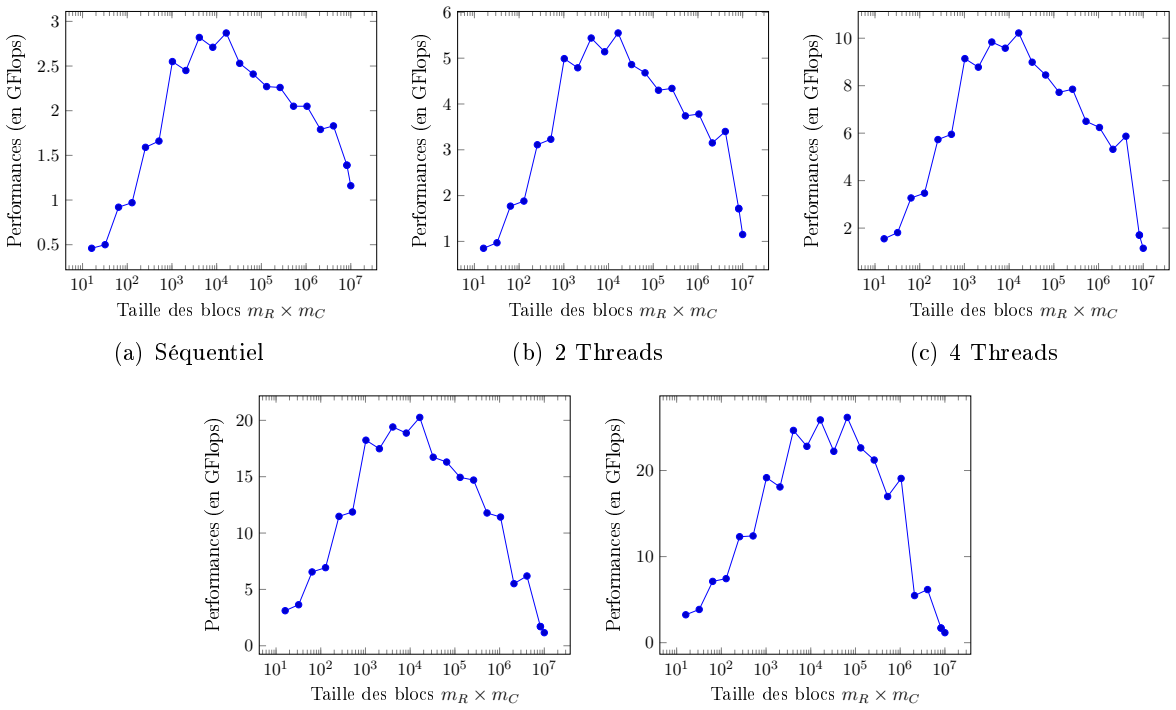
La figure 8.7 montre l'évolution des performances en GFlops en fonction de la taille de la tuile en simple et double précision sur COMP1. Nous avons testé n_T=1 000 et n_T=10 000 car il n'y a pas de découpage en blocs de lignes et donc il est possible que les tuiles n'entrent pas dans la mémoire GPU pour n_T=10 000. Nous constatons qu'en double précision les performances sont proches quel que soit le nombre de lignes. En revanche, en simple précision, les performances sont bien meilleures pour n_T=10 000. La différence de performances entre la double précision et la simple précision vient de la différence entre le nombre de cœurs CUDA dédiés à la double précision et le nombre de cœurs CUDA dédiés à la simple précision. En double précision, la performance



(a) Séquentiel (b) 2 Threads (c) 4 Threads

(d) 8 Threads (e) 16 Threads

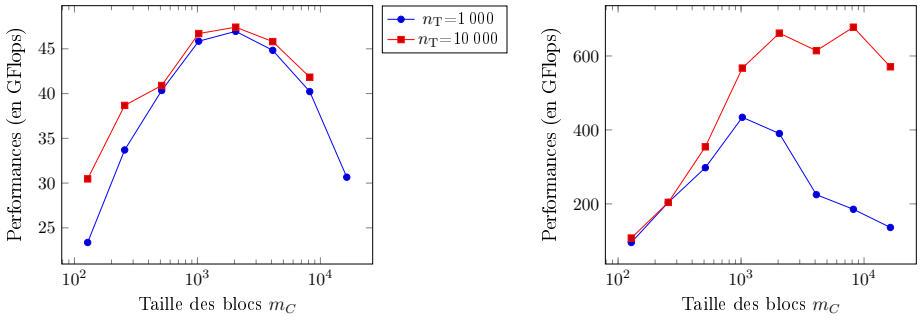
FIGURE 8.5 – ALG_SHARE : Influence de de m_R et m_C sur la performance en double précision sur COMP1.



(a) Séquentiel (b) 2 Threads (c) 4 Threads

(d) 8 Threads (e) 16 Threads

FIGURE 8.6 – ALG_SHARE : Influence de de m_R et m_C sur la performance en simple précision sur COMP1.

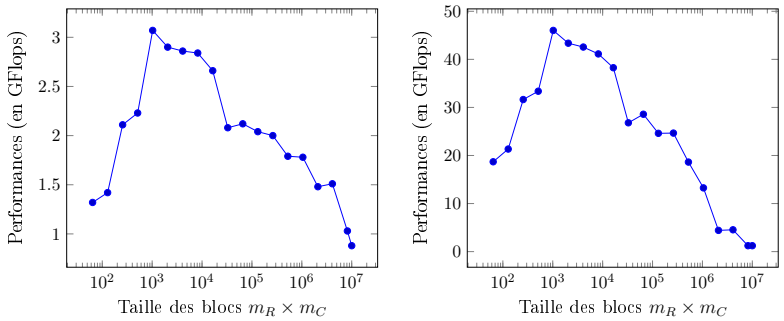


(a) Double précision (b) Simple précision

FIGURE 8.7 – ALG_GPU : Influence de $m_C$sur les performances (en GFlops) sur COMP1 pour n_V=30 000.

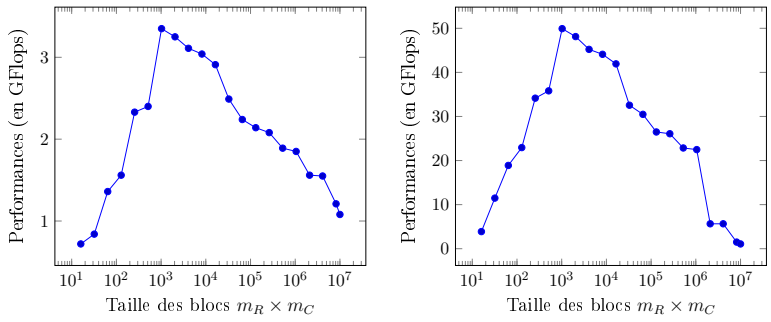
optimale est atteinte pour $m_C{}^* = 2\,048$. En simple précision, il faut choisir $m_C{}^* = 1\,024$ pour n_T=1 000 et $m_C{}^* = 8\,192$ pour $n_T = 10\,000$ afin d'avoir la meilleure performance.

Les figures 8.8 et 8.9 montrent l'évolution des performances en fonction de la taille des blocs de lignes et de colonnes pour ALG_SHARE sur COMP2 pour un problème de taille n_T=1 000 et n_V=10 000. Nous constatons que la performance optimale est obtenue avec $m_R{}^* = 16$ et $m_C{}^* = 64$ en simple et double précision.



(a) Séquentiel (b) 18 threads

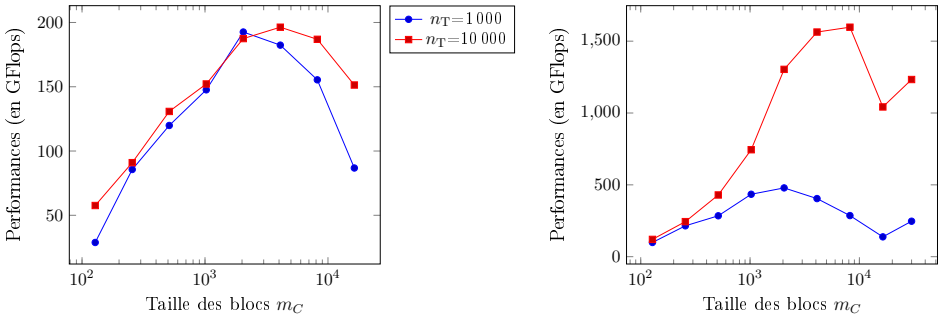
FIGURE 8.8 – Effet de la taille des blocs sur les performances de ALG_SHARE en double précision sur COMP2.



(a) Séquentiel (b) 18 threads

FIGURE 8.9 – Effet de la taille des blocs sur les performances de ALG_SHARE en simple précision sur COMP2.

La figure 8.10 représente l'évolution des performances de ALG_GPU sur COMP3 en fonction



(a) Double précision (b) Simple précision

FIGURE 8.10 – ALG_GPU : Influence de m_C sur les performances (en GFlops) sur COMP2 pour n_V=30 000.

	Nombre de threads	m_R*	m_C*
ALG_SHARE	1 à 8	64	256
ALG_SHARE	16	128	512
ALG_GPU	N.R.	N.R.	2 048

(a) Double précision

	n_T	m_R*	m_C*
ALG_SHARE	-	64	256
ALG_GPU	1 000	N.R.	1 024
ALG_GPU	10 000	N.R.	8 192

(b) Simple précision

TABLE 8.7 – Paramètres (m_R*,m_C*) optimaux sur COMP1
- : $n_T \in \{1\,000, 10\,000\}$; N.R : non requis

	m_R*	m_C*
ALG_SHARE	16	64
ALG_GPU	N.R.	2 048

(a) Double précision

	n_T	m_R*	m_C*
ALG_SHARE	-	16	64
ALG_GPU	1 000	N.R.	2 048
ALG_GPU	10 000	N.R.	8 192

(b) Simple précision

TABLE 8.8 – Paramètres (m_R*,m_C*) optimaux sur COMP2 et COMP3
- : $n_T \in \{1\,000, 10\,000\}$; N.R : non requis

de la taille de la tuile. En double précision, nous constatons que les performances sont assez similaires pour n_T=1 000 et n_T=10 000. En simple précision, les performances sont supérieures pour n_T=10 000. Nous retrouvons en revanche que les performances en simple précision sont plus élevées que celles en double précision, car le nombre de cœurs CUDA simple précision est plus grand que le nombre de cœurs CUDA double précision. En double précision, les performances optimales sont atteintes pour m_C* = 2 048. En simple précision, nous obtenant les meilleures performances pour n_T=1 000 en prenant m_C* = 2 048 et pour n_T=10 000 en prenant m_C* = 8 192.

La table 8.7 récapitule les paramètres optimaux en simple et double précision pour ALG_SHARE et ALG_GPU sur COMP1. La table 8.8 reprend les paramètres optimaux en simple et double précision pour ALG_SHARE sur COMP2 et pour ALG_GPU sur COMP3.

8.2.4 Comparaison des performances des algorithmes

Pour comparer les performances, nous prenons les tailles de tuiles données en Table 8.7 et Table 8.8. Pour ALG_DIST, la matrice de données est découpée en $n_P$tuiles réparties sur chaque processeur et chaque processeur utilise ALG_SHARE avec les paramètres (m_R*,m_C) donnant les performances optimales en séquentiel. Nous rappelons que le temps de référence est le temps séquentiel obtenu avec ALG_SHARE. Pour ALG_HYBR, la profondeur de la boucle (c'est-à-dire le nombre de noyaux CUDA lancés simultanément) a été choisie égale au nombre de *threads* utilisés.

La table 8.9 montre les accélérations obtenues en double précision avec ALG_SHARE sur COMP1. La table 8.11 montre les accélérations en simple précision pour COMP1 pour ALG_SHARE. Dans les deux cas, les accélérations obtenues sont quasi linéaires avec une accélération autour de 11 au lieu de 16 pour seize *threads*. Les tables 8.10 et 8.12 donnent les performances obtenues avec ALG_SHARE en double et simple précision et montrent que les meilleures performances sont obtenues avec seize *threads*. La table 8.13 montre les accélérations obtenues sur COMP1 avec ALG_GPU en simple et double précision. Nous constatons que les accélérations sont meilleures lorsque le nombre de lignes augmente. En double précision, ALG_GPU permet d'obtenir une accélération entre 13 et 15 donc supérieure à celle obtenue avec ALG_SHARE avec 16 *threads*. En simple précision, les accélérations de ALG_GPU sont nettement supérieures à celles obtenues avec ALG_SHARE. Nous remarquons qu'en simple précision, les accélérations sont bien plus grandes qu'en double précision grâce au nombre supérieur de cœurs CUDA en simple précision. En simple et double précision, les performances sont meilleures pour n_T=10 000. Les tables 8.14 et 8.16 montrent les accélérations obtenues avec ALG_HYBR en double et simple précision sur COMP1. Nous remarquons que les accélérations augmentent avec le nombre de *threads*. En revanche, le gain en double précision est assez faible entre 2 *threads* et 16 *threads*. Cela s'explique sans doute par le faible nombre de cœurs CUDA dédiés à la double précision. La table 8.15 confirme que les meilleures performances sont atteintes avec seize *threads* avec n_T=10 000. En simple précision, le gain est meilleur lorsque le nombre de *threads* (et donc le nombre de noyaux CUDA lancés) augmente jusqu'à huit *threads*. Avec seize *threads*, les accélérations sont moins bonnes qu'avec huit *threads* en simple précision : on peut supposer que les seize copies ralentissent ALG_HYBR. La table 8.17 confirme que les meilleures performances sont atteintes avec huit *threads* en particulier pour le plus gros problème de taille 10 00 par 100 000.

Les figures 8.11 et 8.12 comparent les accélérations et performances obtenues avec les trois algorithmes sur COMP1 en double et simple précision. Nous retrouvons que l'accélération de ALG_SHARE est quasi-linéaire en particulier en simple précision. Nous constatons également que ALG_HYBR est plus performant que ALG_GPU à partir de huit *threads*, excepté pour le problème de très grande taille (10 000 par 100 000) en simple précision. Bien que ne partant pas de l'algorithme le plus performant, ALG_HYBR surpasse ALG_GPU en permettant à plus de noyaux CUDA de s'exécuter en parallèle.

		Temps CPU	Accélération			
		Sequentiel	2 threads	4 threads	8 threads	16 threads
1 000	10 000	35,39	**1,71**	**3,21**	**6,43**	**10,93**
	50 000	860,28	**1,73**	**3,14**	**6,21**	**10,79**
	100 000	3461,98	**1,72**	**3,13**	**6,14**	**10,60**
10 000	10 000	346,09	**1,77**	**3,24**	**6,48**	**11,24**
	50 000	8275,41	**1,73**	**3,16**	**6,30**	**10,86**
	100 000	28046,13	**1,40**	**2,66**	**5,30**	**9,10**

TABLE 8.9 – ALG_SHARE : Temps de référence (en secondes) et accélérations (en gras) en double précision sur COMP1.

		Performance				
		Séquentiel	2 threads	4 threads	8 threads	16 threads
1 000	10000	2,83	4,84	9,08	18,17	*30,87*
	50 000	2,91	5,01	9,14	18,06	*31,36*
	100 000	2,89	4,98	9,05	17,73	*30,63*
10 000	10 000	2,89	5,12	9,35	18,72	*32,46*
	50 000	3,02	5,22	9,55	19,04	*32,82*
	100 000	3,57	4,99	9,48	18,90	*32,44*

TABLE 8.10 – ALG_SHARE : Performances (en GFlops) en double précision sur COMP1 .

La table 8.18 montre les accélérations obtenues avec ALG_SHARE en double précision sur COMP2. Nous constatons que l'accélération est éloigné du cas idéal avec une accélération de 38 pour 64 *threads*, obtenant une efficacité de 0,60. La table 8.19 montre les performances de

		Temps CPU	Accélération			
		Sequentiel	2 threads	4 threads	8 threads	16 threads
1 000	10 000	31,49	**1,94**	**3,70**	**7,40**	**10,39**
	50 000	788,48	**1,94**	**3,70**	**7,33**	**10,27**
	100 000	3190,54	**1,96**	**3,69**	**7,21**	**9,99**
10 000	10 000	308,66	**1,95**	**3,72**	**7,42**	**10,52**
	50 000	7685,21	**1,96**	**3,72**	**7,44**	**10,51**
	100 000	30769,54	**1,96**	**3,72**	**7,42**	**10,48**

TABLE 8.11 – ALG_SHARE : Temps de référence (en secondes) et accélérations (en gras) en simple précision sur COMP1.

		Performance				
		Séquentiel	2 threads	4 threads	8 threads	16 threads
1 000	10 000	3,18	6,16	11,75	23,49	*33,00*
	50 000	3,17	6,14	11,73	23,24	*32,55*
	100 000	3,13	6,14	11,58	22,60	*31,32*
10 00	10 000	3,24	6,30	12,06	24,04	*34,07*
	50 000	3,25	6,37	12,09	24,20	*34,18*
	100 000	3,25	6,37	12,10	24,11	*34,05*

TABLE 8.12 – ALG_SHARE : Performances (en GFlops) en simple précision sur COMP1 .

		Temps CPU	ALG_GPU	
		Sequentiel	Accélération	Performances
1 000	10 000	35,39	**12,09**	34,15
	50 000	860,28	**13,77**	40,00
	100 000	3 461,98	**13,98**	40,39
10 000	10 000	346,09	**14,55**	42,03
	50 000	8 275,41	**15,50**	*46,83*
	100 000	28 046,13	**13,26**	*47,27*

(a) Double précision

		Temps CPU	ALG_GPU	
		Sequentiel	Accélération	Performances
1000	10000	31,49	**37,31**	118,47
	50000	788,48	**58,51**	185,51
	100000	3190,54	**59,99**	188,01
1000	10000	308,66	**119,36**	386,66
	50000	7685,21	**146,15**	*475,43*
	100000	30769,54	**150,08**	*487,74*

(b) Simple précision

TABLE 8.13 – ALG_GPU : Accélérations (en gras) par rapport au temps de référence (en secondes) en double et simple précision sur COMP1.

		Temps CPU	Accélération			
		Sequentiel	2 threads	4 threads	8 threads	16 threads
1 000	10 000	35,39	**10,34**	**11,44**	**12,88**	**13,23**
	50 000	860,28	**12,44**	**14,14**	**15,50**	**16,24**
	100 000	3 461,98	**12,64**	**14,29**	**15,57**	**16,47**
10 000	10 000	346,09	**12,36**	**14,18**	**15,22**	**15,37**
	50 000	8 275,41	**13,00**	**14,97**	**16,12**	**16,78**
	100 000	28 046,13	**11,13**	**12,81**	**13,80**	**14,38**

TABLE 8.14 – ALG_HYBR : Accélérations (en gras) par rapport au temps de référence en double précision sur COMP1.

		Performances			
		2 threads	4 threads	8 threads	16 threads
1 000	10 000	32,31	36,39	37,37	30,87
	50 000	36,14	41,09	45,04	47,19
	100 000	36,51	41,27	44,98	47,57
10 000	10 000	35,71	40,98	43,97	44,41
	50 000	39,29	45,23	48,71	*50,68*
	100 000	39,70	45,69	49,19	*51,26*

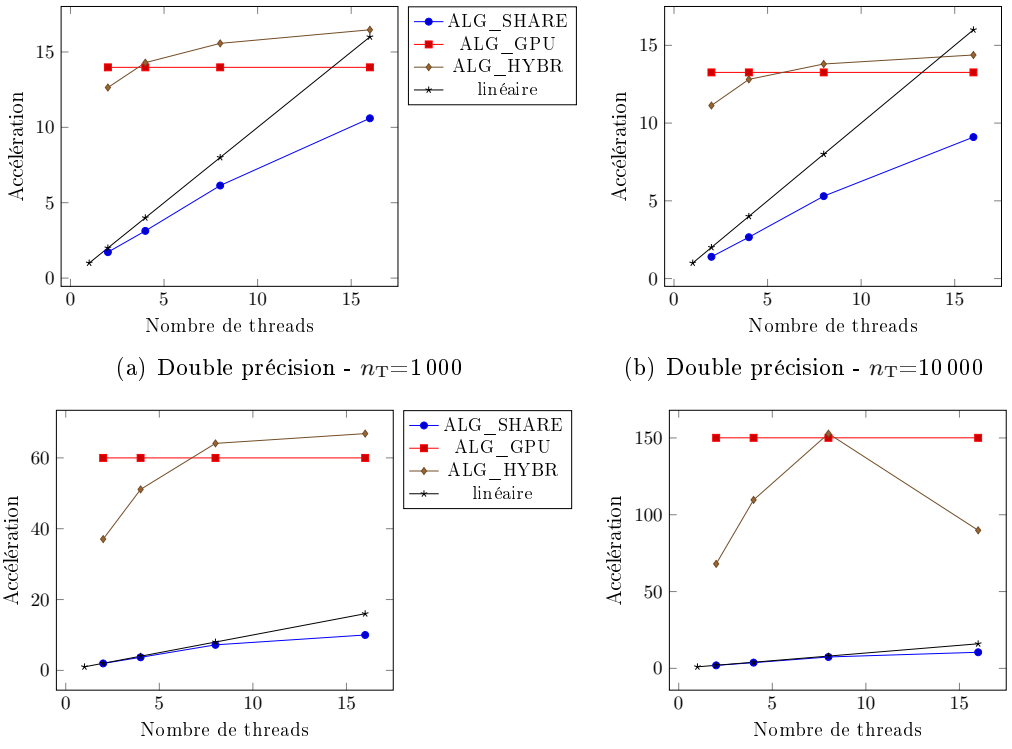
TABLE 8.15 – ALG_HYBR : Performances (en GFlops) en double précision sur COMP1.

		Temps CPU	Accélération			
		Sequentiel	2 threads	4 threads	8 threads	16 threads
1 000	10 000	31,49	**27,33**	**41,32**	**52,48**	**49,98**
	50 000	788,48	**40,13**	**64,69**	**80,85**	**85,05**
	100 000	3 190,54	**37,07**	**51,10**	**64,09**	**66,85**
10 000	10 000	308,66	**58,94**	**93,36**	**109,73**	**75,91**
	50 000	7685,21	**68,68**	**112,68**	**130,60**	**91,96**
	100 000	30769,54	**68,01**	**109,65**	**152,82**	**89,87**

TABLE 8.16 – ALG_HYBR : Accélérations (en gras) par rapport au temps de référence en simple précision sur COMP1.

		Performances			
		2 threads	4 threads	8 threads	16 threads
1 000	10 000	86,80	131,22	166,65	158,71
	50 000	127,23	205,10	256,35	269,65
	100 000	116,17	160,15	200,86	209,52
10 000	10 000	190,93	302,45	355,46	245,92
	50 000	223,42	366,53	424,83	299,13
	100 000	221,03	356,37	*496,65*	292,06

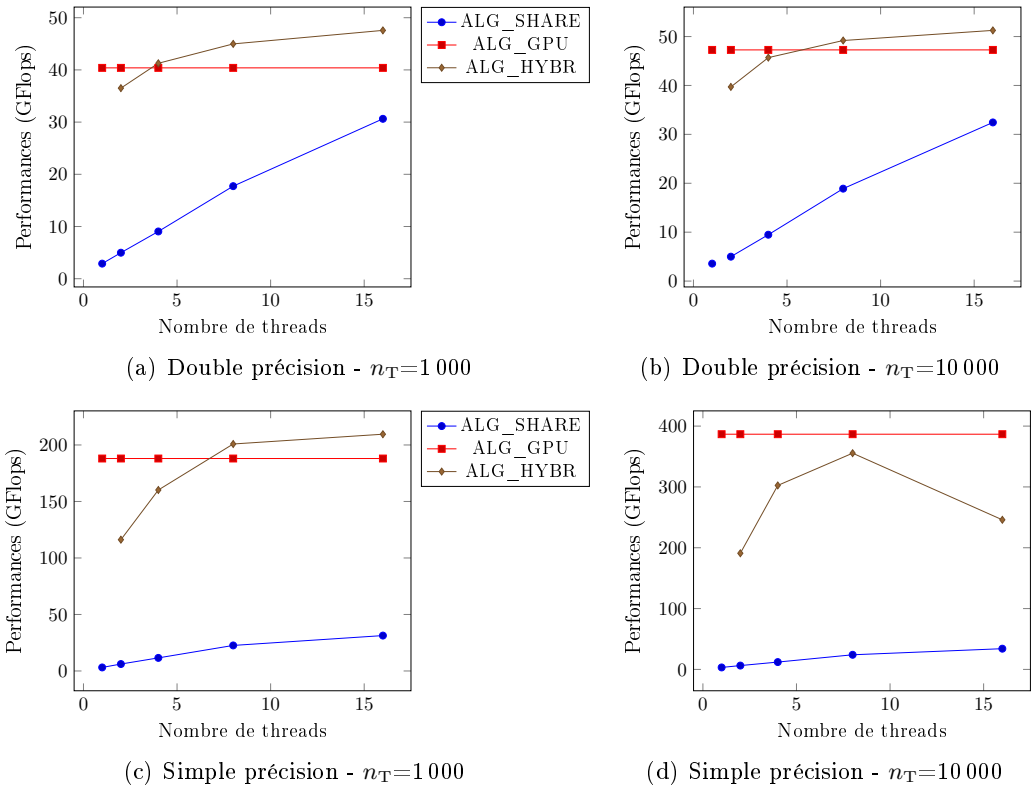
TABLE 8.17 – ALG_HYBR : Performances (en GFlops) en simple précision sur COMP1.



(a) Double précision - n_T=1 000 (b) Double précision - n_T=10 000

(c) Simple précision - n_T=1 000 (d) Simple précision - n_T=10 000

FIGURE 8.11 – Comparaison des accélérations obtenues avec ALG_SHARE, ALG_GPU et ALG_HYBR sur COMP1.



(a) Double précision - n_T=1 000 (b) Double précision - n_T=10 000

(c) Simple précision - n_T=1 000 (d) Simple précision - n_T=10 000

FIGURE 8.12 – Comparaison des performances (en GFlops) obtenues avec ALG_SHARE, ALG_GPU et ALG_HYBR sur COMP1.

ALG_SHARE sur COMP2 en double précision et confirme que les meilleures performances sont atteintes avec 64 *threads*. La table 8.20 montre les accélérations obtenues sur COMP2 en simple précision. En simple précision, les accélérations se rapprochent du cas idéal avec une accélération entre 53 et 58 pour 64 *threads*, ce qui donne une efficacité d'environ 0,8. La performance optimale en simple précision est supérieure à celle obtenue en double précision (186 GFlops en simple précision contre 116 GFlops en double précision) mais reste atteinte pour 64 *threads*.

		Temps CPU	Accélération					
n_T	n_V	Séquentiel	2 threads	4 threads	8 threads	16 threads	32 threads	64 threads
1 000	32 000	337,49	**1,46**	**2,79**	**5,70**	**10,78**	**20,98**	**40,38**
	64 000	1 350,27	**1,44**	**2,75**	**5,51**	**10,38**	**20,12**	**38,76**
	128 000	5 411,50	**1,42**	**2,74**	**5,44**	**10,23**	**19,71**	**38,14**
10 000	32 000	3 232,96	**1,33**	**2,51**	**5,09**	**9,62**	**18,69**	**36,60**
	64 000	12 927,66	**1,32**	**2,51**	**5,09**	**9,63**	**18,71**	**36,66**
	128 000	52 044,22	**1,31**	**2,51**	**5,09**	**9,69**	**18,83**	**36,76**

TABLE 8.18 – ALG_SHARE : Temps de calcul (en secondes) et accélération (en gras) en double précision sur COMP2.

		Temps CPU	Accélération					
n_T	n_V	Séquentiel	2 threadsThread	4 threads	8 threads	16 threads	32 threads	64 threads
1 000	32 000	3,03	4,42	8,45	17,29	32,71	63,64	122,51
	64 000	3,03	4,36	8,33	16,70	31,50	61,02	117,58
	128 000	3,03	4,30	8,30	16,47	30,98	59,68	115,48
10 000	32 000	3,17	4,21	7,94	16,12	30,45	59,18	*115,92*
	64 000	3,17	4,19	7,95	16,12	30,52	59,26	*116,14*
	128 000	3,15	4,12	7,91	16,03	30,50	59,27	*115,71*

TABLE 8.19 – ALG_SHARE : Performances (en GFlops) en double précision sur COMP2.

		Temps CPU	Accélération					
n_T	n_V	Séquentiel	2 threads	4 threads	8 threads	16 threads	32 threads	64 threads
1 000	32 000	313,40	**1,90**	**3,72**	**7,61**	**14,01**	**26,87**	**52,59**
	64 000	1 258,86	**1,89**	**3,64**	**7,41**	**13,66**	**26,40**	**51,44**
	128 000	5 042,33	**1,88**	**3,70**	**7,27**	**13,48**	**25,83**	**49,80**
1 000	32 000	2 948,54	**1,93**	**3,84**	**7,64**	**13,99**	**27,04**	**53,54**
	64 000	11 785,00	**1,93**	**3,83**	**7,63**	**14,00**	**27,09**	**53,56**
	128 000	51 957,14	**2,01**	**4,20**	**8,28**	**15,36**	**29,79**	**58,31**

TABLE 8.20 – ALG_SHARE : Temps de calcul (en secondes) et accélérations (en gras) en simple précision sur COMP2.

		Temps CPU	Accélération					
n_T	n_V	Séquentiel	2 threadsThread	4 threads	8 threads	16 threads	32 threads	64 threads
1 000	32 000	3,27	6,20	12,17	24,88	45,76	87,78	171,84
	64 000	3,25	6,16	11,84	24,11	44,46	85,89	167,37
	128 000	3,25	6,11	12,01	23,61	43,80	83,94	161,83
1 000	32 000	3,47	6,71	13,34	26,54	48,58	93,91	*185,95*
	64 000	3,48	6,72	13,30	26,53	48,65	94,14	*186,15*
	128 000	3,15	6,34	13,23	26,10	48,44	93,94	*183,86*

TABLE 8.21 – ALG_SHARE : Performances (en GFlops) en simple précision sur COMP2.

La table 8.22 montre les temps de calcul et les performances de ALG_GPU en simple et double précision sur COMP3.

Si on compare les performances en double précision de ALG_GPU à celles obtenues avec ALG_SHARE sur COMP2 (**cf. Table 8.19**), on constate que les performances de ALG_GPU se situent entre les performances de ALG_SHARE pour 32 *threads* et celles obtenues pour 64 *threads*. En revanche, en simple précision, on obtient une performance maximale pour ALG_GPU de 192 GFlops contre une performance de 171 GFlops pour ALG_SHARE (**cf. Table 8.20**) pour n_T=1 000 et une performance de 1 326 GFlops pour n_T=10 000 bien supérieure à celle obtenue avec ALG_SHARE.

n_T	n_V	Temps CPU	Performance
1 000	32 000	10,40	98,44
	64 000	38,51	*106,37*
	128 000	180,21	90,92
10 000	32 000	56,44	181,42
	64 000	210,74	194,36
	128 000	OOM	

(a) Double précision

n_T	n_V	Temps CPU	Performance
1 000	32 000	5,98	171,15
	64 000	21,72	188,61
	128 000	84,92	192,94
10 000	32 000	10,32	991,74
	64 000	33,13	1 236,40
	128 000	123,50	*1 326,58*

(b) Simple précision

TABLE 8.22 – ALG_GPU : Temps de calcul (en secondes) et performances (en GFlops) sur COMP3 OOM : dépassement mémoire

La table 8.23 montre les temps d'exécution, les performances (en GFlops) et les accélérations obtenues pour ALG_DIST sur COMP2 en double précision. Nous constatons que pour 2 processeurs, l'accélération est presque inexistante. L'accélération maximale est autour de 34 et est atteinte avec 64 processeurs. Si on compare les performances de ALG_DIST avec celles de ALG_SHARE (**cf. Table 8.19**), on constate que ALG_SHARE est plus performant que ALG_DIST avec une performance maximale de 102 GFlops pour ALG_DIST contre une performance de 115 GFlops pour ALG_SHARE en prenant n_T=1 000 en double précision. Le résultat n'est pas étonnant puisque ALG_DIST est ralenti par les échanges de messages. La table 8.24 montre les résultats de ALG_DIST sur COMP2 en simple précision. Comparés aux accélérations et aux performances (**cf. Table 8.23**) en double précision, on constate que les accélérations et performances en simple précision sont plus élevées qu'en double précision : on obtient une per-

formance maximale de 127 GFlops en simple précision contre 101 GFlops en double précision. Mais cette performance reste inférieure à la performance obtenue avec ALG_SHARE qui est de 171 GFlops (**cf. Table 8.21**). En revanche pour n_T=10 000, nous constatons que ALG_DIST surpasse les performances de ALG_SHARE à partir de 16 *threads* en simple précision, ce qui donne une performance de 194 GFlops avec 64 *threads* pour ALG_DIST contre une performance de 184 GFlops pour ALG_SHARE. De même en double précision pour n_T=10 000, à partir de 8 *threads*, ALG_DIST est plus performant que ALG_SHARE avec une performance de 157 GFlops pour ALG_DIST contre une performance de 115 GFlops pour ALG_SHARE. Une explication possible est qu'avec n_T=10 000, les blocs sont suffisamment gros pour que le temps de calcul recouvre le temps de communication ; autrement dit, lorsque le calcul est fini, la communication est achevée donc le processeur n'a pas besoin d'attendre.

La figure 8.13 compare les performances de ALG_SHARE sur COMP2, ALG_GPU sur COMP3 et ALG_DIST sur COMP2 en double et simple précision pour n_T=1 000. Elle confirme que ALG_SHARE est plus performant que ALG_DIST lorsque le nombre de *threads* est suffisamment élevé. Quant à ALG_GPU, cette figure montre qu'il est plus performant en simple précision alors qu'en double précision ALG_SHARE est plus performant en prenant 64 *threads*.

n_T=1 000	n_V=64 000			n_V=256 000		
	Temps	Accélération	Performance	Temps	Accélération	Performance
1 processeur	1 373,27		2,98	22 137,35		2,96
2 processeurs	1 281,88	**1,07**	3,20			
4 processeurs	491,84	**2,79**	8,33	8 056,59	**2,75**	8,13
8 processeurs	214,53	**6,40**	19,09	3 494,47	**6,33**	18,75
16 processeurs	113,57	**12,09**	36,06	1 802,66	**12,28**	36,36
32 processeurs	64,77	**21,20**	63,24	1 028,87	**21,52**	63,70
64 processeurs	40,40	**33,99**	*101,38*	646,94	**34,22**	*101,30*

(a) n_T=1 000

n_V=128 000	n_T=1 000			n_T=5 000		n_V=10 000	
	Temps	Accélération	Perf.	Temps	Perf.	Temps	Perf.
1 processeur	5 453,23		3,00				
4 processeurs	1 957,46	**2,79**	8,37	9 553,56	8,57		
8 processeurs	857,86	**6,36**	19,10	4 071,46	20,12	7 925,92	20,67
16 processeurs	455,16	**11,98**	36,00	1 991,19	41,14	3 947,95	41,50
32 processeurs	254,91	**21,39**	64,27	1 021,80	80,17	2 037,57	80,41
64 processeurs	160,118	**34,06**	*102,32*	560,29	*146,21*	1 044,87	*156,80*

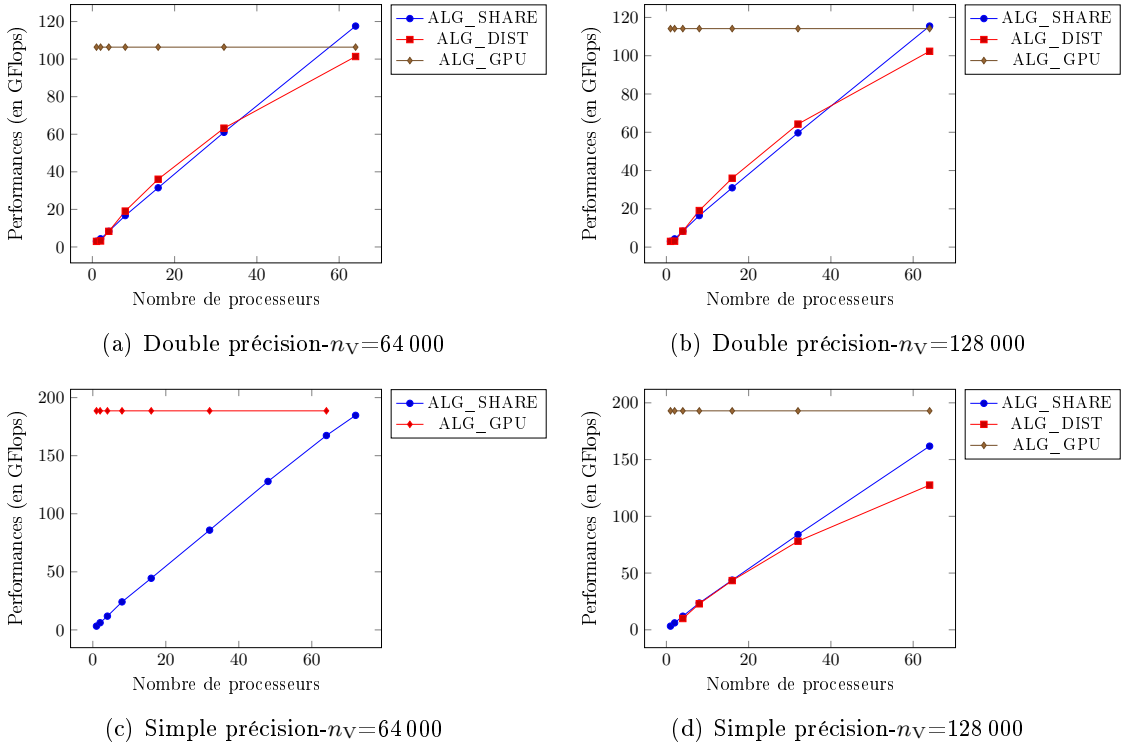
(b) n_V=128 000

TABLE 8.23 – ALG_DIST : Temps de calcul (en secondes), accélération (en gras) et performance (en GFlops) en double précision sur COMP2.

n_V=128 000	n_T=1 000		n_T=5 000		n_V=10 000	
	Temps	Performance	Temps	Performance	Temps	Performance
4 processeurs	1 646,935	9,95	7 852,281	10,43	15 500,902	10,57
8 processeurs	715,106	22,91	3 327,252	24,62	6 554,438	25,00
16 processeurs	377,659	43,38	1 661,936	49,29	3 268,145	50,13
32 processeurs	209,852	78,07	836,477	97,93	1 623,331	100,93
64 processeurs	128,49	*127,51*	441,224	*185,66*	844,68	*193,97*

TABLE 8.24 – ALG_DIST : Temps de calcul (en secondes), accélérations (en gras) et performances (en GFlops) en simple précision pour n_V=128 000 sur COMP2.

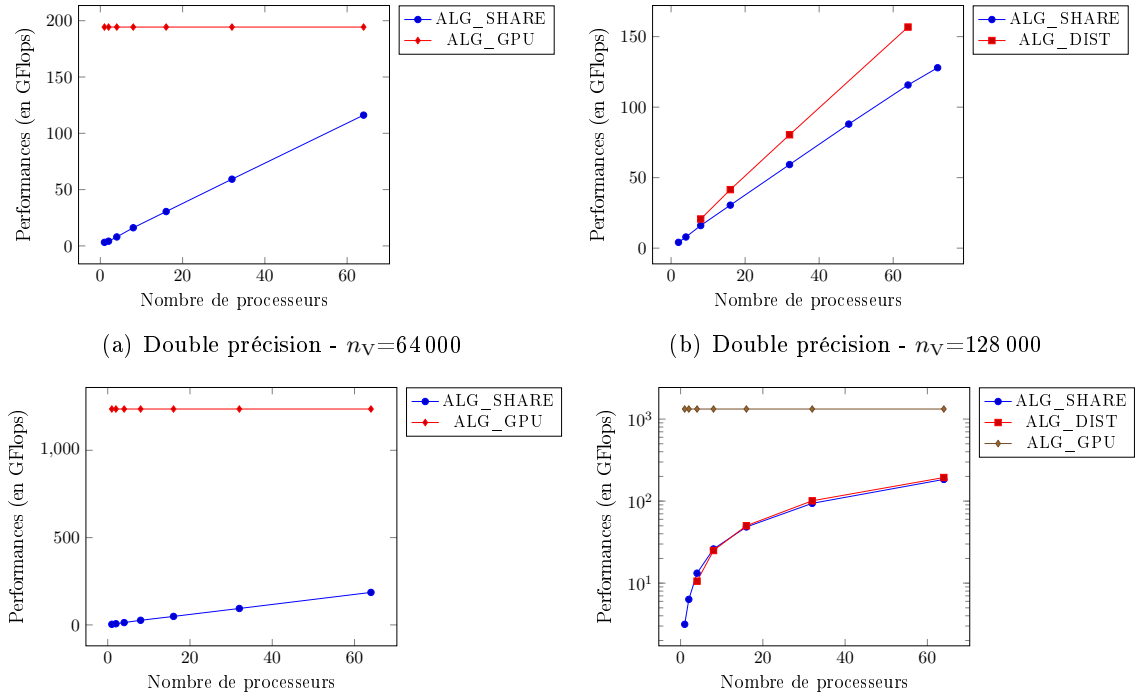
Après comparaison des quatre algorithmes sur des environnements différents, nous pouvons conclure que ALG_SHARE est plus performant en double précision



(a) Double précision-n_V=64 000 (b) Double précision-n_V=128 000

(c) Simple précision-n_V=64 000 (d) Simple précision-n_V=128 000

FIGURE 8.13 – Comparaison de ALG_SHARE (sur COMP2), ALG_GPU (sur COMP3) et ALG_DIST (sur COMP2) pour n_T=1 000.



(a) Double précision - n_V=64 000 (b) Double précision - n_V=128 000

(c) Simple précision - n_V=64 000 (d) Simple précision - n_V=128 000

FIGURE 8.14 – Comparaison de ALG_SHARE (sur COMP2), ALG_GPU (sur COMP3) et ALG_DIST (sur COMP2) pour n_T=10 000.

que ALG_GPU et ALG_DIST lorsque le nombre de *threads* employés est suffisant et le nombre de lignes peu élevé. Si le nombre de lignes devient élevé, ALG_DIST pourra être préféré à ALG_SHARE. Sur un poste de travail avec peu de cœurs disponibles, ALG_GPU pourra être préférée à ALG_SHARE. En revanche, en simple précision, ALG_HYBR ou ALG_GPU sont plus performants.

8.3 Parallélisation de l'algorithme de Ward

Nous comparons les algorithmes de parallélisation suivants :

ALG_WSHARE l'algorithme de Ward utilisant la parallélisation de la distance décrite en section 7.5.2.1 (découpage en blocs de lignes et de colonnes et parcours des tuiles avec une seule boucle) et la parallélisation des itérations décrite en section 6.3.3,

ALG_WGPU l'algorithme de Ward utilisant la parallélisation de la distance décrite en section 7.3.2 (c'est-à-dire un découpage en tuiles de colonnes et un parcours des tuiles avec une double boucle) sans parallélisation des itérations.

Les tables 8.25 et 8.27 montrent les accélérations obtenues pour ALG_WSHARE sur les itérations de l'algorithme (sans le calcul de la distance). Nous constatons que les accélérations sont faibles, montant jusqu'à 2,36 en double précision avec 16 *threads* et jusqu'à 2 en simple précision. Les tables 8.26 et 8.28 montrent les accélérations obtenues pour les itérations sur COMP2 avec ALG_WSHARE. Nous constatons que même avec un plus grand nombre de *threads*, l'accélération est seulement de 3 en double précision et de 2,4 en simple précision. Cela n'est pas étonnant puisque seules des sous-étapes sont parallélisées. De plus, les étapes de mises à jour ont une parallélisation peu efficace à cause du test pour avoir si une parcelle est encore active, et donc les mises à jour effectuées se réduisent au fur et à mesure des itérations. Les accélérations varient peu lorsque le nombre de lignes augmentes, ce qui est normal puisque le nombre de lignes n'intervient pas dans les itérations.

		Temps CPU	Accélération			
n_T	n_V	Séquentiel	2 threads	4 threads	8 threads	16 threads
1 000	32 000	131,18	**1,16**	**1,55**	**2,11**	**2,11**
	64 000	839,98	**1,15**	**1,51**	**2,02**	**2,35**
	128 000	4 431,97	**1,15**	**1,46**	**1,88**	**2,23**
10 000	32 000	141,77	**1,16**	**1,56**	**2,22**	**2,16**
	64 000	897,16	**1,15**	**1,52**	**2,02**	**2,36**
	128 000	4 608,40	**1,09**	**1,32**	**1,60**	**1,82**

TABLE 8.25 – ALG_WSHARE : Temps (en secondes) et accélérations (en gras) pour les itérations en double précision sur COMP1.

		Temps CPU	Accélération					
n_T	n_V	Séquentiel	2 threads	4 threads	8 threads	16 threads	32 threads	64 threads
1 000	32 000	147,36	**0,95**	**1,16**	**1,48**	**1,91**	**2,43**	**2,98**
	64 000	630,53	**1,03**	**1,18**	**1,43**	**1,85**	**2,36**	**2,93**
	128 000	2 694,18	**1,07**	**1,24**	**1,51**	**1,90**	**2,42**	**3,01**
10 000	32 000	158,18	**0,98**	**1,18**	**1,49**	**1,87**	**2,47**	**3,08**
	64 000	682,03	**1,04**	**1,18**	**1,46**	**1,87**	**2,44**	**3,02**
	128 000	2 919,58	**1,07**	**1,24**	**1,54**	**1,95**	**2,49**	**3,11**

TABLE 8.26 – ALG_WSHARE : Temps (en secondes) et accélérations (en gras) pour les itérations en double précision sur COMP2.

Les tables 8.29 et 8.30 montrent les accélérations pour l'algorithme complet. En double précision, nous obtenons des accélérations entre 4 et 5 pour n_T=1 000 et entre 8 et 10 pour n_T=10 000. Les accélérations en simple précision sont moindres avec une accélération maximale de 4,13 pour

		Temps CPU	Accélération			
n_T	n_V	Séquentiel	2 threads	4 threads	8 threads	16 threads
1 000	32 000	111,36	**0,98**	**1,13**	**1,49**	**1,40**
	64 000	667,15	**1,04**	**1,24**	**1,62**	**1,93**
	128 000	3 890,86	**1,05**	**1,24**	**1,51**	**1,84**
10 000	32 000	122,16	**1,02**	**1,24**	**1,55**	**1,65**
	64 000	706,26	**1,04**	**1,31**	**1,54**	**1,77**
	128 000	4 129,35	**1,08**	**1,28**	**1,56**	**1,70**

TABLE 8.27 – ALG_WSHARE : Temps (en secondes) et accélérations (en gras) pour les itérations en simple précision sur COMP1.

		Temps CPU	Accélération					
n_T	n_V	Séquentiel	2 threads	4 threads	8 threads	16 threads	32 threads	64 threads
1 000	32 000	142,51	**0,89**	**1,05**	**1,43**	**1,59**	**1,97**	**2,31**
	64 000	615,71	**0,95**	**1,08**	**1,25**	**1,55**	**2,16**	**2,27**
	128 000	2627,37	**1,02**	**1,15**	**1,33**	**1,79**	**2,18**	**2,30**
10 000	32 000	153,72	**0,90**	**1,07**	**1,27**	**1,61**	**2,23**	**2,42**
	64 000	667,17	**0,97**	**1,06**	**1,28**	**1,60**	**2,03**	**2,35**
	128 000	2855,16	**0,95**	**1,16**	**1,38**	**1,68**	**2,30**	**2,43**

TABLE 8.28 – ALG_WSHARE : Temps (en secondes) et accélérations (en gras) pour les itérations en simple précision sur COMP2.

n_T=1 000 et de 8,83 pour n_T=10 000. Les tables 8.31 et 8.32 montrent les accélérations obtenues sur COMP2 avec ALG_WSHARE pour l'algorithme complet. Nous trouvons jusqu'à 16 *threads* des accélérations proches de celles observées sur COMP1. Nous constatons cependant qu'en double précision, l'accélération est autour de 8 pour n_T=1 000 et autour de 24 pour n_V=10 000 avec 64 *threads*. En simple précision, sur COMP2, l'accélération maximale est de 6,75 pour n_T=1 000 (légèrement inférieure à celle obtenue en double précision) et de 26,18 pour n_T=10 000 avec 64 *threads*.

		Temps CPU	Accélération			
n_T	n_V	Séquentiel	2 threads	4 threads	8 threads	16 threads
1 000	32 000	503,57	**1,53**	**2,56**	**4,24**	**5,29**
	64 000	2 315,11	**1,45**	**2,30**	**3,59**	**4,71**
	128 000	10 144,33	**1,38**	**2,09**	**3,09**	**4,00**
10 000	32 000	3 678,83	**1,67**	**3,17**	**6,12**	**9,74**
	64 000	14 912,60	**1,66**	**3,06**	**5,75**	**9,14**
	128 000	62 504,73	**1,63**	**2,93**	**5,32**	**8,23**

TABLE 8.29 – ALG_WSHARE : Temps (en secondes) et accélérations (en gras) en double précision sur COMP1.

		Temps CPU	Accélération			
n_T	n_V	Séquentiel	2 threads	4 threads	8 threads	16 threads
1 000	32 000	432,68	**1,55**	**2,33**	**3,66**	**3,88**
	64 000	1 960,38	**1,50**	**2,21**	**3,32**	**4,13**
	128 000	9 095,98	**1,42**	**1,99**	**2,75**	**3,45**
10 000	32 000	3 300,12	**1,87**	**3,50**	**6,57**	**8,83**
	64 000	12 723,46	**1,76**	**3,25**	**5,90**	**7,95**
	128 000	52 426,44	**1,82**	**3,13**	**5,55**	**7,23**

TABLE 8.30 – ALG_WSHARE : Temps (en secondes) et accélérations (en gras) en simple précision sur COMP1.

		Temps CPU	Accélération					
n_T	n_V	Séquentiel	2 threads	4 threads	8 threads	16 threads	32 threads	64 threads
1 000	32 000	484,86	**1,26**	**1,95**	**3,05**	**4,47**	**6,32**	**8,38**
	64 000	1 980,85	**1,28**	**1,93**	**2,89**	**4,21**	**5,93**	**7,92**
	128 000	8 105,87	**1,28**	**1,96**	**2,92**	**4,17**	**5,83**	**7,82**
10 000	32 000	3 391,15	**1,31**	**2,38**	**4,57**	**8,06**	**14,30**	**24,28**
	64 000	13 609,74	**1,31**	**2,37**	**4,53**	**7,97**	**14,02**	**23,51**
	128 000	54 963,98	**1,29**	**2,38**	**4,53**	**8,00**	**13,96**	**23,35**

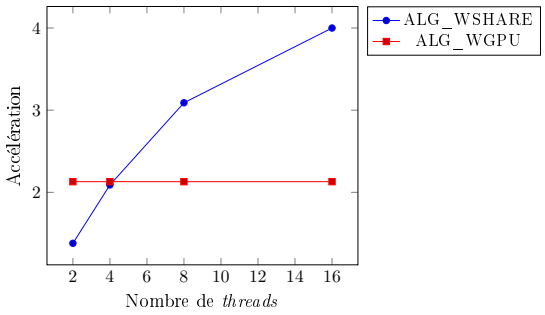
TABLE 8.31 – ALG_WSHARE : Temps (en secondes) et accélérations (en gras) en double précision sur COMP2.

		Temps CPU	Accélération					
n_T	n_V	Séquentiel	2 threads	4 threads	8 threads	16 threads	32 threads	64 threads
1 000	32 000	455,92	**1,40**	**2,07**	**3,24**	**4,07**	**5,44**	**6,75**
	64 000	1 874,59	**1,43**	**2,05**	**2,83**	**3,84**	**5,63**	**6,35**
	128 000	7 669,79	**1,46**	**2,10**	**2,88**	**4,16**	**5,49**	**6,17**
10 000	32 000	3 102,27	**1,83**	**3,40**	**6,12**	10,13	17,43	**26,18**
	64 000	12 452,20	**1,83**	**3,36**	**6,03**	**9,90**	**16,29**	**24,68**
	128 000	54 812,39	**1,90**	**3,69**	**6,56**	**10,78**	**18,36**	**26,52**

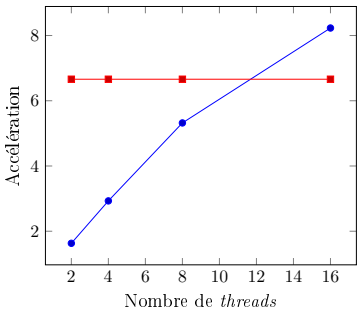
TABLE 8.32 – ALG_WSHARE : Temps (en secondes) et accélérations (en gras) en simple précision sur COMP2.

La table 8.33 montre les accélérations obtenues avec ALG_WGPU sur COMP1 en simple et double précision. Pour n_T=1 000, en simple et double précision les accélérations obtenues avec ALG_WSHARE sur COMP1 se situent entre les accélérations de ALG_WSHARE obtenues pour 4 *threads* et celles obtenues avec 8 *threads*. Pour n_T=10 000, en double précision les accélérations de ALG_WGPU sont situées entre les accélérations de ALG_WSHARE obtenues pour 8 *threads* et celles pour 16 *threads* sur COMP1. En revanche pour n_T=10 000 en simple précision, les accélérations de ALG_WGPU surpassent celles obtenues avec ALG_WSHARE à cause des accélérations très élevées du calcul de la distance qui permettent de compenser la non parallélisation des itérations sur COMP1. La table 8.34 montre les accélérations obtenues sur COMP3 avec ALG_WGPU par rapport à l'exécution en séquentiel de ALG_WSHARE sur COMP3. Nous constatons que pour n_T=1 000 en double précision, les accélérations de ALG_WSHARE sont équivalentes aux accélérations obtenues avec 16 *threads* pour ALG_SHARE sur COMP2. Pour n_T=10 000 en double précision, elles se situent entre les accélérations de ALG_WSHARE obtenues pour 32 *threads* et pour 64 *threads*. En simple précision pour n_T=1 000, les accélérations obtenues avec ALG_WGPU sur COMP3 sont équivalentes à celles obtenues sur COMP2 avec ALG_WSHARE pour 16 *threads*. En revanche pour n_T=10 000 en simple précision, les accélérations de ALG_WGPU sur COMP3 sont équivalentes à celles de ALG_WSHARE sur COMP2 obtenues avec 64 *threads*.

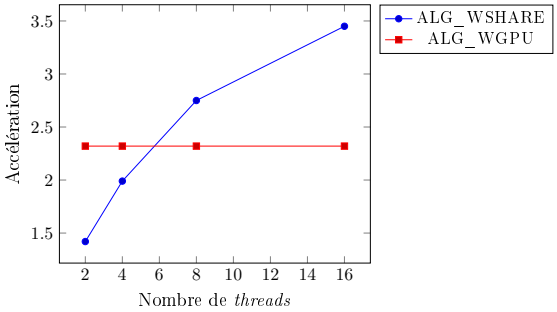
La figure 8.15 compare les accélérations obtenues avec ALG_WSHARE et ALG_WGPU sur COMP1 et confirme qu'**excepté le cas de la simple précision et n_T=10 000, ALG_WSHARE est plus performant que ALG_WGPU.** Cependant, comme ALG_WGPU atteint au moins les performances de ALG_WSHARE avec quatre *threads*, nous pouvons penser qu'un algorithme hybride qui utiliserait ALG_GPU pour le calcul de la distance et la parallélisation des itérations avec OpenMP dépasserait les performances de ALG_WSHARE. La figure 8.16 compare les accélérations obtenues avec ALG_WSHARE et ALG_WGPU sur COMP2 et sur COMP3 (pour ALG_WGPU) : nous retrouvons qu'en double précision et en simple précision pour n_T=1 000, ALG_WSHARE surpasse ALG_WGPU. En simple précision pour n_T=10 000, ALG_WSHARE atteint la performance de ALG_WGPU avec 64 *threads*.



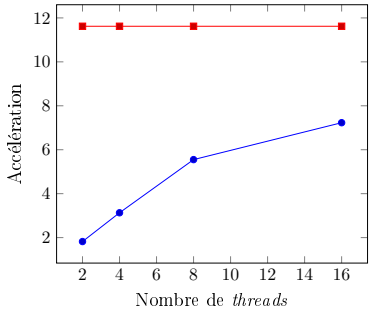
(a) Double précision-n_T=1 000



(b) Double précision-n_T=10 000

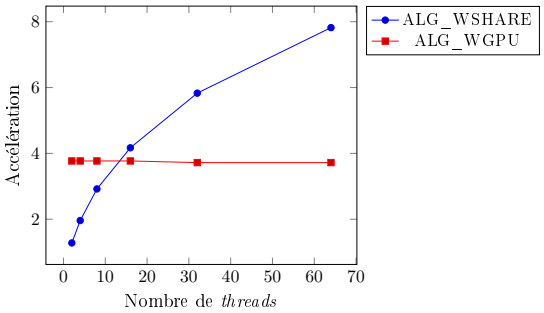


(c) Simple précision-n_T=1 000

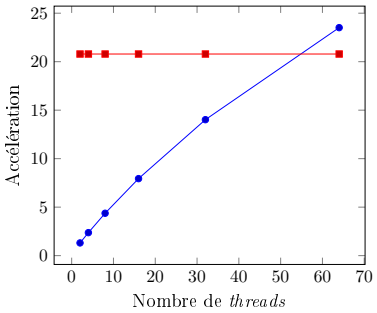


(d) Simple précision-n_T=10 000

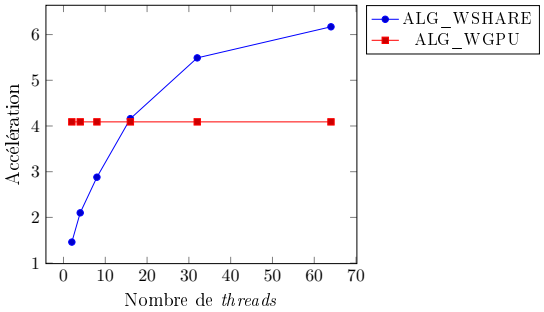
FIGURE 8.15 – Comparaison de ALG_WSHARE et ALG_WGPU en double et simple précision sur COMP1 pour n_V=128 000.



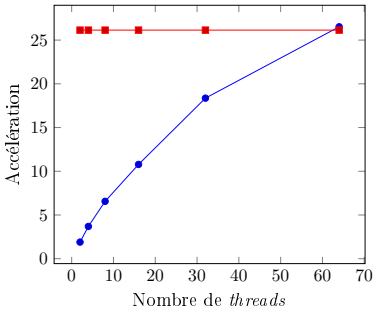
(a) Double précision-n_T=1 000 et n_V=128 000



(b) Double précision-n_T=10 000 et n_V=64 000



(c) Simple précision-n_T=1 000 et n_V=128 000



(d) Simple-n_T=10 000 et n_V=128 000

FIGURE 8.16 – Comparaison de ALG_WSHARE (sur COMP2) et ALG_WGPU (sur COMP3) en double précision .

		Temps CPU	Accélération
n_T	n_V	Séquentiel	CUDA
1 000	32 000	503,57	**3,20**
	64 000	2 315,11	**2,45**
	128 000	10 144,33	**2,13**
10 000	32 000	3 678,83	**10,13**
	64 000	14 912,60	**8,40**
	128 000	62 504,73	**6,66**

(a) Double précision

		Temps CPU	Accélération
n_T	n_V	Séquentiel	CUDA
1 000	32 000	432,68	**3,56**
	64 000	1 960,38	**2,84**
	128 000	9 095,98	**2,32**
10 000	32 000	3 300,12	**22,96**
	64 000	12 723,46	**15,82**
	128 000	52 426,44	**11,62**

(b) Simple précision

TABLE 8.33 – ALG_WGPU : Temps (en secondes) et accélérations (en gras) en double et simple précision sur COMP1 .

		Temps CPU	Accélération	
n_T	n_V	Séquentiel	/ COMP3	/ COMP2
1 000	32 000	454,53	**4,21**	**4,49**
	64 000	1 835,61	**4,09**	**4,42**
	128 000	8 014,27	**3,72**	**3,77**
10 000	32 000	3 318,07	**20,59**	**21,06**
	64 000	13 294,26	**20,31**	**20,79**
	128 000	OOM	OOM	OOM

(a) Double précision

		Temps CPU	Accélération	
n_T	n_V	Séquentiel	/ COMP3	/ COMP2
1 000	32 000	425,61	**4,03**	**4,32**
	64 000	1 730,83	**3,88**	**4,21**
	128 000	7 010,41	**3,74**	**4,09**
10 000	32 000	3 165,52	**26,74**	**26,20**
	64 000	12 623,50	**25,26**	**24,92**
	128 000	50 626,4258	**24,14**	**26,14**

(b) Simple précision

TABLE 8.34 – ALG_WGPU : Temps (en secondes) et accélérations (en gras) en double et simple précision sur COMP3 .
OOM : dépassement mémoire

8.4 Discussion

Nous avons montré qu'en séquentiel, ALGF (c'est-à-dire le découpage en tuiles de la matrice de données) est le plus performant. Nous avons ensuite comparé les parallélisations de la distance. Sur un serveur de calcul disposant d'au moins une cinquantaine de cœurs (comme COMP2), ALG_SHARE est le plus performant en double précision. En simple précision, ALG_GPU a de meilleures performances que les autres algorithmes. Cette différence entre la simple et double précision s'explique par un nombre de *streaming multiprocessors* dédiés à la simple précision plus important que le nombre de *streaming multiprocessors* dédiés à la double précision sur les GPUs NVIDIA. Sur un ordinateur de bureau (comme COMP1), ALG_GPU est le plus adapté car ses performances sont équivalentes à celle de ALG_SHARE en utilisant tous les cœurs. Un défaut de ALG_GPU est que les tuiles sont copiées sur le GPU avec toutes les lignes de la matrice $\boldsymbol{X}$, ce qui pourrait être limitant si le nombre de lignes est très élevé. Toutefois, sur COMP1 en double précision, l'algorithme est exécutable avec n_T=100 000 et n_V=100 000 ; et avec n_T=60 000 et n_V=128 000.

Concernant l'algorithme de Ward, ALG_WSHARE est plus rapide que ALG_WGPU excepté en simple précision lorsque le nombre de lignes est élevé. Nous pourrions de plus combiner la parallélisation de la distance sur GPU avec la parallélisation des itérations avec OpenMP. Même si le gain sur les itérations est faible (réduction du temps d'un facteur 2), la combinaison des deux parallélisations devrait être meilleure que ALG_WSHARE. Cette amélioration apparaît d'autant

plus importante qu'une fois la distance parallélisée, le temps d'exécution des itérations domine le temps de l'algorithme. Un inconvénient de notre approche est sa consommation mémoire car il faut stocker le vecteur de distances. Par exemple pour traiter un sujet, la parcellisation des données d'IRMf dans la substance grise requiert environ 250 Gio.

Chapitre 9

Parcellisation des données d'IRMf

Ce chapitre décrit les résultats de la parcellisation obtenus sur les données issues du jeu de données présenté au chapitre 5 après application de la censure. Dans la première section, nous montrons la parcellisation du signal de pression en fonction des répétitions. La deuxième section étudie l'efficacité de l'algorithme de Ward sur des simulations. Enfin, nous montrons les résultats de la parcellisation sur notre jeu de données.

9.1 Parcellisation du signal de pression

Un signal de pression relevé sur l'épiglotte de chaque sujet a été mesuré pendant les sessions d'IRMf. L'objectif était double :

1. mieux connaître la façon dont le sujet gère un stimulus liquide dans l'IRM ;
2. établir un lien entre les mouvements de tête et les mouvements particuliers induits par ce type de stimuli.

Le protocole mis en œuvre a été conçu pour maximiser la puissance statistique au moyen d'un grand nombre de stimulations liquides (80) et d'un délai prolongé post-stimulation pour revenir à la ligne de base. Outre les données d'IRM, les relevés de pression constituaient un jeu de données de choix pour étudier la déglutition des sujets placés en décubitus dorsal. C'est pourquoi nous avons parcellisé le signal de pression en fonction des répétitions. Nous avons séparé le signal de pression selon les 80 stimulations et nous avons parcellisé les stimulations avec un algorithme de Ward en déterminant le nombre de parcelles optimal selon la *slope statistic* comme introduit dans le chapitre 6.

9.1.1 Analyse intra-individuelle du signal de pression

Nous avons caractérisé la variabilité entre les 80 stimulations pour chaque sujet selon différents critères : l'amplitude des pics de déglutition et leur retard par rapport à l'ordre de déglutition. Pour cela, nous avons estimé à l'aide de la fonction FINDPEAKS de MATLAB le nombre de pics au-dessus de 10% de variation du signal, leur amplitude et leur retard par rapport à l'ordre de déglutition de chaque stimulation. Cette variation de signal est calculée pour chaque stimulation séparément selon la formule

$$psct(t) = \frac{s(t) - \overline{s}}{\overline{s}} \ ,$$

le calcul de la moyenne se faisant sur toute la durée de la stimulation. Afin de gérer les différences de durée de la période de repos entre deux stimulations, la parcellisation des 80 stimulations ne considère pour chaque stimulation que les quinze premières secondes de la période de repos.

Pour ne pas alourdir le manuscrit, nous ne présentons ici que les résultats obtenus avec le sujet n^{o}1. Les analyses du signal de pression pour les autres sujets sont données à l'annexe E.2.

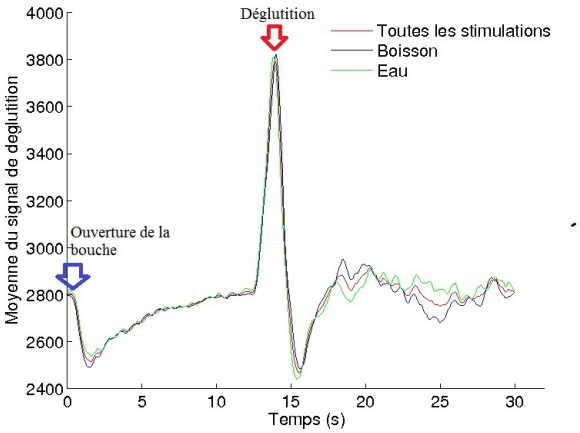


FIGURE 9.1 – Moyenne des signaux de déglutition sur toutes les stimulations pour le sujet 1.

La figure 9.1 représente la moyenne sur toutes les stimulations pour le sujet 1. On peut y voir un premier pic qui représente l'ouverture de la bouche (flèche bleue) puis un second pic (flèche rouge) représentant la déglutition. On remarque qu'il n'y a presque pas de différences entre les moyennes sur les stimulations de boisson sucrée et les stimulations d'eau pure. Cependant, moyenner les stimulations efface la présence de plusieurs pics de déglutitions qui se retrouvent dans plusieurs stimulations.Le tableau 9.1 caractérise les pics sur l'ensemble des stimulations. Nous remarquons un pic lors de la période d'avertissement (donc avant l'ordre de déglutition) qui correspond sans doute à l'ouverture de la bouche. Le pic n^{o}2 est le pic de déglutition (l'ordre arrivant à 12 secondes). Les pics suivants ont lieu lors de la phase de repos : ce sont des pics post-déglutition. Nous constatons que les amplitudes de ces pics post-déglutition sont plus faibles que le pic de déglutition mais restent élevées. Le tableau E.1 montre le nombre de pics et leur caractérisation pour chaque stimulation. On peut y voir la variabilité des stimulations : deux stimulations successives ne se ressemblant parfois pas (par exemple les stimulations n^{os}18,19 et 20).

N^o de pic	**Localisation (en s)**	**Amplitude du pic (en %)**
1	0.74±0.65	36.65±11.03
2	13.84±0.27	43.70±2.88
3	19.37±0.99	37.06±10.02
4	23.22±1.58	40.02±7.82
5	28.47±0.95	33.55±12.26
6	32.15±1.45	32.14±11.74

TABLE 9.1 – Caractérisation des pics sur l'ensemble des stimulations pour le sujet 1.

Nous avons ensuite cherché à déterminer s'il existe des schémas communs à ces déglutitions. Pour cela, nous avons parcellisé les répétitions afin d'obtenir des groupes de stimulations et donc des types de déglutition spécifiques. Le critère *slope statistic* est ensuite utilisé pour déterminer le nombre de parcelles, c'est-à-dire le nombre de signaux de déglutition types. Pour ce sujet, le critère *slope statitic* détermine que le nombre optimal de parcelles est 39. La table 9.3 montre la répartition des stimulations dans les différentes parcelles. Nous remarquons que plusieurs parcelles ne comportent qu'une seule stimulation. Les figures 9.2 à 9.8 montrent les variations du signal en pourcentages pour les centroïdes des différentes parcelles, calculés comme moyenne des stimulations dans la parcelle. Nous constatons que le centroïde 24 comporte une forte variation au moment de la déglutition et que les centroïdes 3, 8, 18, 26, 27, 29, 30 et 35 ont trois fortes variations. Le reste des centroïdes présentent deux fortes variations. Le tableau 9.2 montre les pics sur l'ensemble des centroïdes. Il confirme qu'il y a un pic en début de paradigme pour

l'ouverture de la bouche, puis le pic n^o 2 pour le pic de déglutition et ensuite plusieurs pics lors de la phase de repos.

N^o de pic	**Localisation (en s)**	**Amplitude du pic (en %)**
1	0.74±0.65	36.84±10.59
2	13.85± 0.30	43.53±3.40
3	19.49±1.22	35.23±10.82
4	23.72±1.42	38.84± 8.05
5	28.76±0.83	28.56± 9.67

TABLE 9.2 – Caractérisation des pics sur l'ensemble des centroïdes pour le sujet 1.

N^o de la parcelle	**N^{os} des stimulations dans la parcelle**
1	19
2	60
3	11
4	38,74
5	56
6	52,67
7	72
8	7,43
9	12,25,27,71
10	28,76
11	31,42,57,65
12	9,16,58
13	24,39,77
14	21
15	22,33,45
16	14,49
17	2,35,75
18	61
19	34
20	79
21	3,63
22	4
23	6,48
24	18,20,30,36,44,62,66,70,78
25	68
26	80
27	8
28	13,29,59,73
29	41
30	5,40
31	32,54
32	15,17,55
33	69
34	23,47,51,64
35	1
36	26,50
37	37
38	46
39	10,53

TABLE 9.3 – Répartition des stimulations dans les parcelles pour le sujet 1.

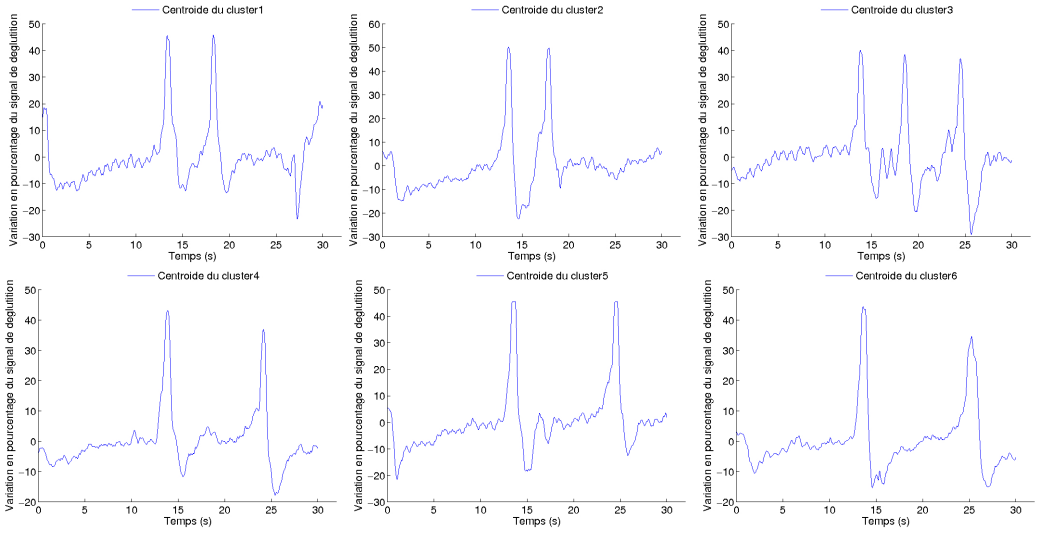


FIGURE 9.2 – Centroïdes des parcelles 1 à 6 pour le sujet 1.

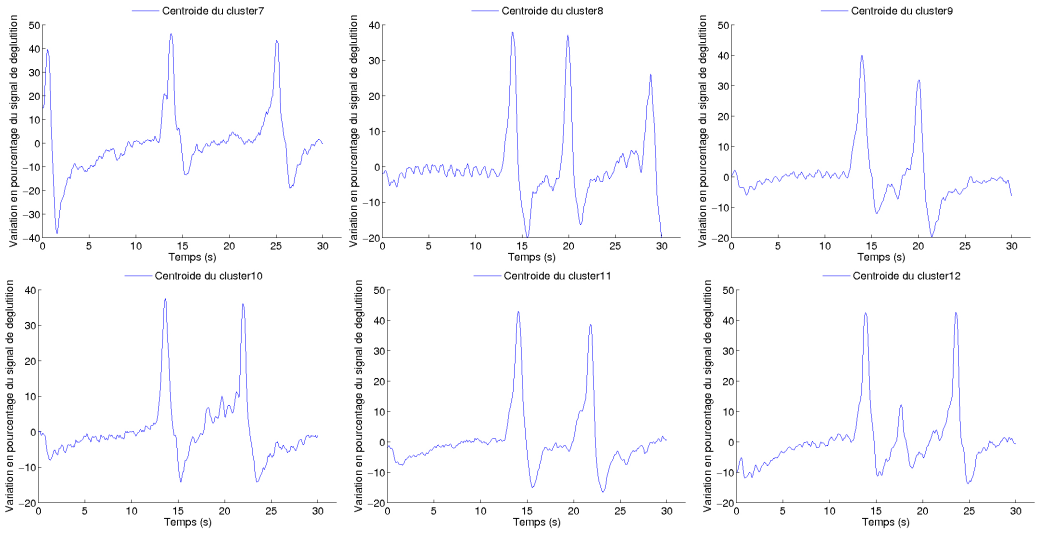


FIGURE 9.3 – Centroïdes des parcelles 7 à 12 pour le sujet 1.

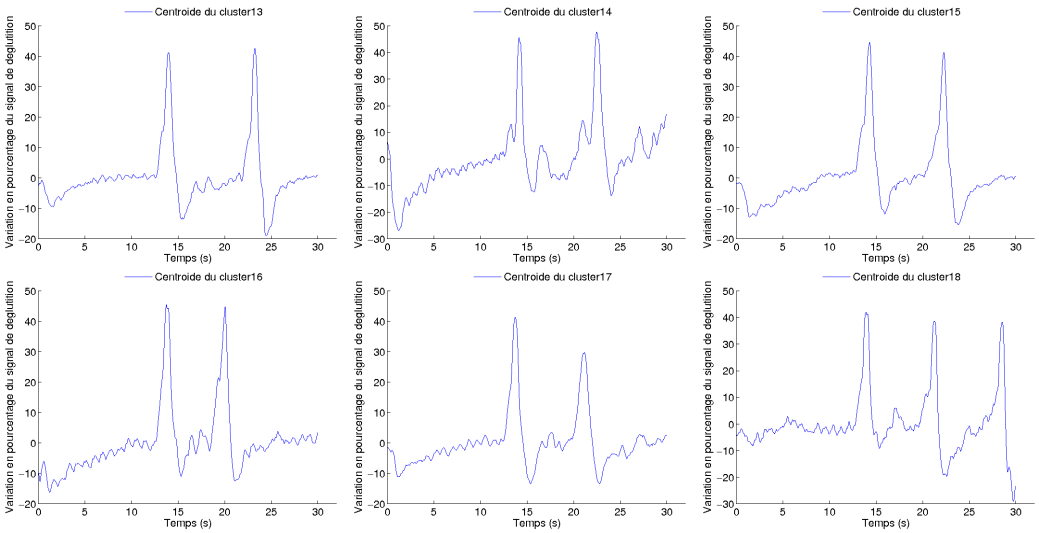


FIGURE 9.4 – Centroïdes des parcelles 13 à 18 pour le sujet 1.

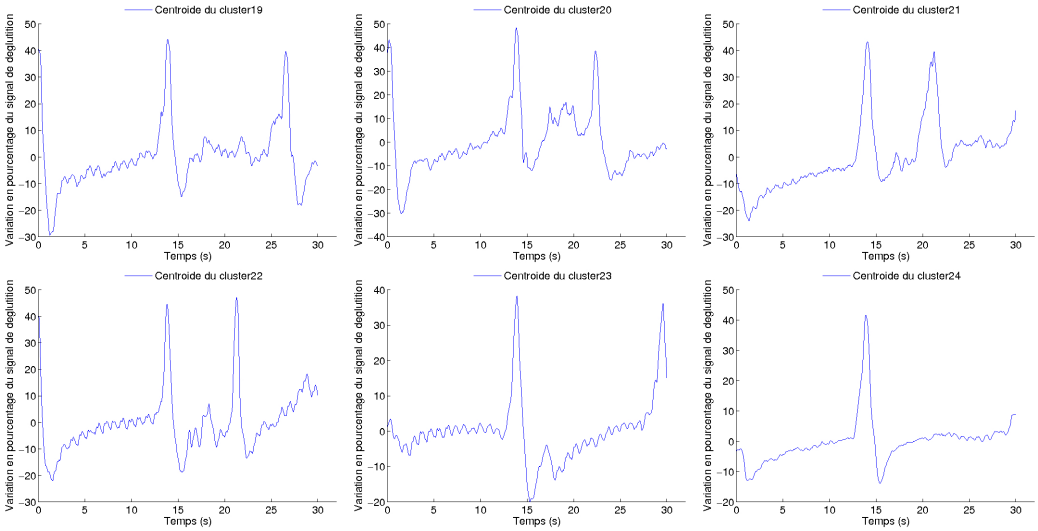


FIGURE 9.5 – Centroïdes des parcelles 19 à 24 pour le sujet 1.

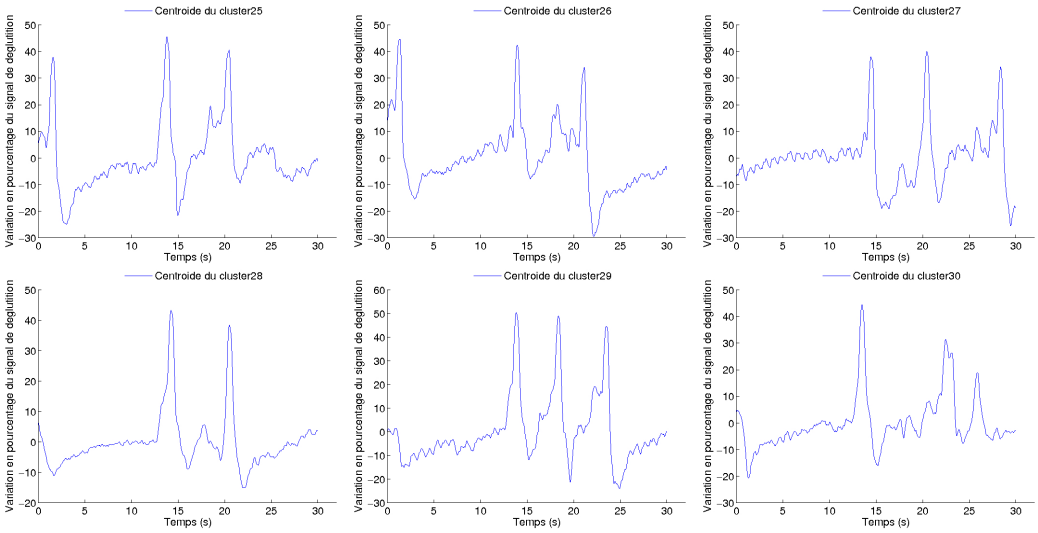


FIGURE 9.6 – Centroïdes des parcelles 25 à 30 pour le sujet 1.

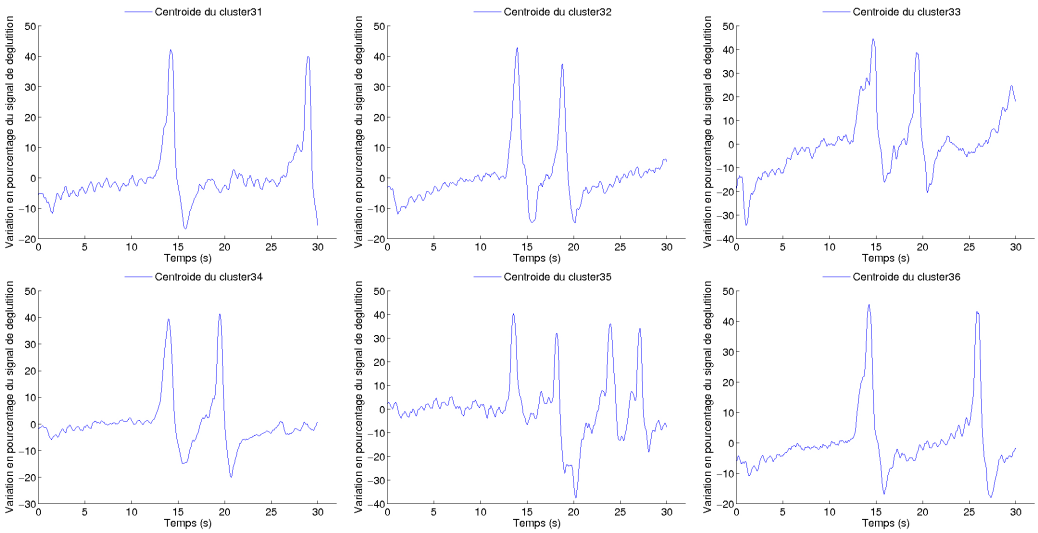


FIGURE 9.7 – Centroïdes des parcelles 31 à 36 pour le sujet 1.

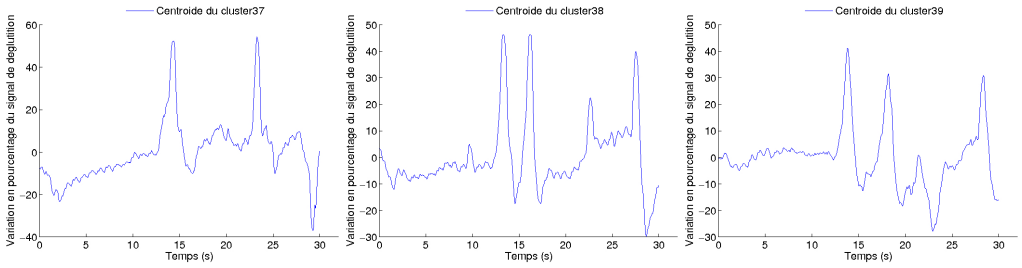
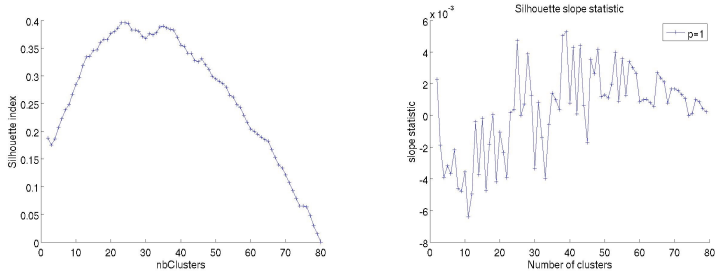


FIGURE 9.8 – Centroïdes des parcelles 37 à 39 pour le sujet 1.



(a) Indice de silhouette (b) *Slope statistic*

FIGURE 9.9 – Détermination du nombre optimal de parcelles pour le sujet 1.

Le nombre de parcelles optimal élevé pour ce sujet montre une forte variabilité intra-individuelle. La figure 9.9(a) montre que l'indice de silhouette (**cf. équation 6.2**) est élevé entre 20 et 40 parcelles sans saut important entre deux valeurs successives (c'est-à-dire entre l'indice de silhouette pour un nombre de parcelles n_C et l'indice de silhouette pour n_C+1). De plus, le deuxième choix du nombre optimal de parcelles avec la *slope statistic* est de 38 (**cf.** Figure 9.9(b) et équation 6.7) et le troisième choix de 25. Cela confirme que les déglutitions réalisées pendant les 80 stimulations sont très diverses et qu'il n'existe pas un seul type de déglutition. La parcellisation permet également de trouver des centroïdes présentant plusieurs pics de variations élevées correspondant à plusieurs déglutitions par le sujet 1.

Après analyse, il apparaît que la déglutition en deux fois est la plus fréquente (28 centroïdes regroupant au total 53 stimulations) pour le sujet 1. La variabilité intra-individuelle est importante, en d'autres termes le sujet ne déglutit jamais de la même façon pendant une session. Nous observons également plusieurs pics post-déglutitions d'amplitude comparable à la première. Ceci suggère que l'analyse postérieure à l'événement « déglutition » du paradigme sera difficile à analyser. De plus, cela confirme l'intérêt de la censure fondée sur le signal. L'analyse des trois autres sujets confirment la variabilité intra-individuelle mais pour les sujets n^{os} 2 et 4, la déglutition en une seule fois est la plus fréquente.

9.1.2 Variabilité inter-sujet

Les observations de la section 9.1.1 en plus de caractériser la variabilité intra-individuelle des stimulations de déglutition permettent de mettre en évidence des différences entre les sujets. Certes, tous les sujets présentent un pic de déglutition. En revanche, les sujets 1 et 3 présentent beaucoup de stimulations comportant plusieurs pics ; alors que pour les sujets 2 et 4, les stimulations n'ont principalement qu'un seul pic au moment de la déglutition. De plus, le sujet 1 a des pics tardifs de déglutition survenant environ six secondes après le premier pic de déglutition et dont les variations du signal sont proches de la variation du premier pic de déglutition ; à l'inverse, le sujet 3 présente un pic de déglutition dès la fin de la phase de déglutition mais les variations

du signal de ces pics tardifs sont plus faibles que la variation du premier pic de déglutition. La parcellisation des stimulations montre également différentes variabilités intra-sujet des stimulations. En effet, pour le sujet 4, le nombre optimal de parcelles n'est que de cinq parcelles contre 37 pour le sujet 1. Les cas du sujet 2 et du sujet 3 sont plus difficiles à classer. En effet, pour le sujet 2, le nombre optimal de parcelles est 2, suggérant une faible variabilité intra-individuelle mais le deuxième maximum donne 37 qui correspond à une variabilité intra-individuelle plus importante. À l'inverse pour le sujet 3, le nombre optimal de parcelles est de 30 dénotant une variabilité intra-individuelle importante mais le deuxième maximum donne une parcellisation en 3, ce qui montrerait une faible variabilité intra-individuelle.

9.1.3 Relation entre le signal de pression et les paramètres de recalage rigide

Nous calculons le coefficient de corrélation de Pearson entre le signal de pression (interpolé toutes les trois secondes sur la durée de l'acquisition) et ces paramètres de réalignement. On s'intéresse également à la corrélation entre le signal de pression et les mesures de mouvement décrit par Power[170] : le *framewise displacement* (FD) , le *translational absolute displacement* (TAD) et le *rotational absolute displacement* (RAD) définis en section 5.3.1. Le tableau 9.4 résume les coefficients de corrélation pour les quatre sujets : on constate qu'il n'y a pas de corrélation linéaire entre le signal de pression et les paramètres de réalignement rigide. **Les paramètres de réalignement ne sont donc pas des régresseurs de non intérêt suffisants pour représenter le mouvement dû à la déglutition traduit par le relevé de pression. Autrement dit, utiliser les régresseurs de non-intérêts conventionnels est une stratégie peu efficace pour prendre en compte les mouvements générés par des stimuli liquides.**

Coefficient de corrélation entre le signal de pression et	**Sujet n^{o}1**	**Sujet n^{o}2**	**Sujet n^{o}3**	**Sujet n^{o}4**
Δx	0.02	0	0.10	-0.02
Δy	0.03	0.06	-0.10	0.06
Δz	0.04	-0.03	-0.06	-0.09
$\Delta roll$	0.02	-0.02	0.08	0.08
$\Delta pitch$	0.0	-0.04	0.06	-0.03
Δyaw	0.04	0.04	0.08	0.02
TAD	-0.04	0.01	-0.08	0.01
RAD	-0.02	-0.02	0.0	-0.07
FD	-0.13	-0.17	-0.18	-0.04

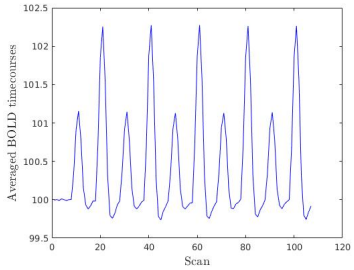
TABLE 9.4 – Coefficient de corrélation entre le signal de pression et les paramètres de mouvement.

9.2 Simulations : l'algorithme de Ward peut-il dissocier les régions de signal des régions de bruit ?

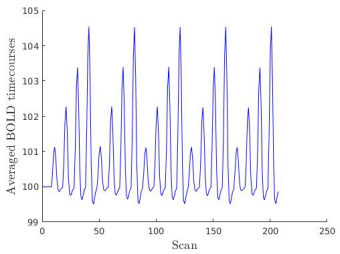
L'algorithme de Ward a d'abord été testé sur des données simulées afin d'évaluer son comportement dans un contexte où le nombre de parcelles est connu. L'objectif est en particulier de déterminer un indice pour estimer le nombre optimal de parcelles.Pour cela, nous avons créé des jeux de données artificiels se limitant à une seule coupe pour tester si l'algorithme de Ward sépare les régions contenant du signal des régions de bruit, si on dispose d'un indice pour calculer le nombre optimal de parcelles et étudier ce qui se passe lorsque le nombre de parcelles est mal estimé.

Pour cela, nous avons créé trois types de signal. Chaque signal peut s'écrire sous la forme $s(t) = bsl + PSC(t) * hrf(t) + \epsilon(t)$. bsl représente la ligne de base fixée à 100. $\epsilon(t)$ est un

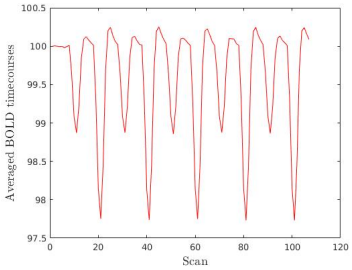
bruit gaussien de moyenne nulle et d'écart-type $\sigma = \frac{bsl}{SNR}$ avec le ratio signal sur bruit $SNR \in \{100, 200, 300\}$. $PSC(t)$ correspond aux variations de signal représentées sous forme d'un signal créneau convolué avec la réponse hémodynamique $hrf(t)$. Le premier type de signal (signal A1 et A2) est composé uniquement de variations positives d'une valeur de 1% pour la première condition, 2% pour la deuxième condition et ainsi de suite. Le deuxième type de signal (signal B1 et B2) correspond à des variations négatives d'une valeur de -1% pour la première condition, -2% pour la deuxième condition et ainsi de suite. Le troisième type de signal est plus hétérogène : le signal appelé C1 est composé de variations positives d'une valeur de 0,5% pour la première condition et 2,5% pour la deuxième condition ; le signal appelé C2 est composé d'une variation positive de 0,5% pour la première condition et d'une variation négative de -2,5% pour la deuxième condition ; et le signal appelé C3 est composé de variations positives d'une valeur de 0,5% pour la première condition, de 2,5% pour la deuxième condition, de 1,5% pour la troisième condition et d'une variation négative de -0,5% pour la quatrième condition. La figure 9.10 montre les différents signaux moyennés dans une région pour atténuer l'effet du bruit avec un SNR de 200. La table 9.5 montre les différentes caractéristiques des simulations utilisées : la configuration utilisée avec éventuellement le nombre de blocs et les signaux employés.



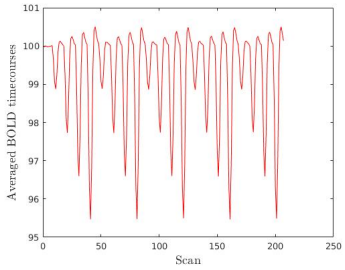
(a) Signal A1- 2 conditions avec variation positive



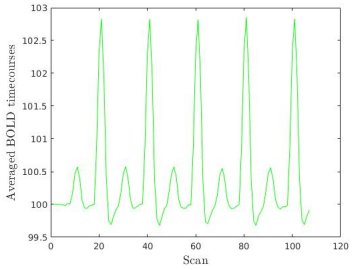
(b) Signal A2- 4 conditions avec variation positive



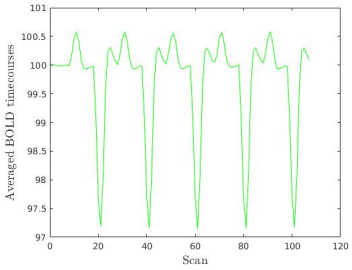
(c) Signal B1- 2 conditions avec variation négative



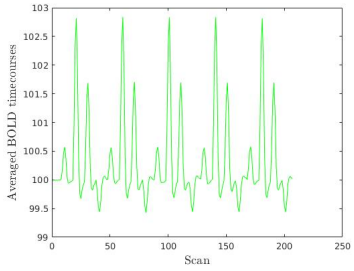
(d) Signal B2- 4 conditions avec variation négative



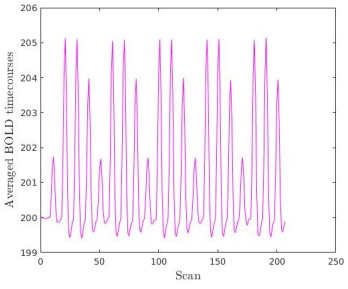
(e) Signal C1- 2 conditions avec variation positive



(f) Signal C2- 2 conditions avec variation positive et négative



(g) Signal C3- 4 conditions avec variation positive et négative



(h) Signal A2+C3- 4 conditions

FIGURE 9.10 – Les différents signaux simulés.

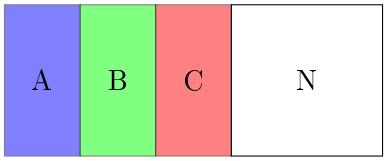
Simulation	Configuration	Signaux utilisés	n_C^*
1	1 (5 blocs)	A1 (2 conditions) et bruit	2
2	1 (5 blocs)	A1 (2 conditions), B1 (2 conditions) et bruit	3
3	1 (5 blocs)	A1 (2 conditions), B1 (2 conditions), C1 (2 conditions) et bruit	4
4	1 (5 blocs)	A1 (2 conditions), B1 (2 conditions), C2 (2 conditions) et bruit	4
5	1 (5 blocs)	A2 (4 conditions) et bruit	2
6	1 (5 blocs)	A2 (4 conditions), B2 (4 conditions) et bruit	3
7	1 (5 blocs)	A2 (4 conditions), B2 (4 conditions), C3 (4 conditions) et bruit	4
8	1 (2 blocs)	A2 (4 conditions) et B2 (4 conditions)	2
9	1 (4 blocs)	A2 (4 conditions), B2 (4 conditions) et bruit	3
10	1 (2 blocs)	A2 (4 conditions) et C2 (4 conditions)	2
11	1 (4 blocs)	A2 (4 conditions), C2 (4 conditions) et bruit	3
12	2	A2 (4 conditions) et bruit	2
13	5	A2 (4 conditions), B2 (4 conditions) et bruit	3
14	6	A2 (4 conditions), B2 (4 conditions) et C3 (4 conditions)	3
15	3	A2 (4 conditions) et bruit	2
16	7	A2 (4 conditions) et C3 (4 conditions)	4
17	4	A2 (4 conditions), B2 (4 conditions), C3 (4 conditions) et bruit	4
18	2	A2 (4 conditions) et B2 (4 conditions)	2
19	2	A2 (4 conditions) et C3 (4 conditions)	2

TABLE 9.5 – Caractéristiques des simulations.

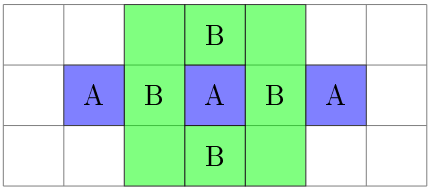
Nous avons ensuite créé différentes configurations des régions. Dans la première configuration, une région est un bloc de 32 par 64 pixels : on accole ensuite entre deux et cinq régions et les régions contenant du signal sont placées à gauche. Dans la deuxième configuration, les régions sont des blocs de 32 par 32 pixels, les régions contenant du signal ne sont pas contiguës et sont au milieu des régions ne contenant que du bruit. La troisième configuration est constituée de 56 blocs de 16 par 16 pixels dont un seul contient du signal. La quatrième configuration comporte 40 blocs de taille soit 4 par 4 pixels soit 4 par 8 pixels soit 8 par 8 pixels avec trois blocs contenant un signal différent. La cinquième configuration ressemble à la configuration 3 excepté que la région centrale contenant le signal A est entourée de huit régions contenant le signal B. La sixième configuration consiste en 25 blocs de 32 par 32 pixels : le bloc au centre contient le signal A, la couronne interne contient le signal B et la couronne externe contient le signal C. La septième et dernière configuration est composée de 15 blocs de 32 par 32 pixels : quatre blocs contiennent le signal A et quatre blocs contiennent le signal C ; et les signaux A et C sont superposés dans un bloc. Les différentes configurations sont illustrées en figure 9.11. Les configurations 1, 2, 3, 4 et 5 ont pour but de déterminer si l'algorithme de Ward peut dissocier les régions de signal des régions de bruit quel que soit l'agencement et la taille des régions. Les configurations 1 (sans région de bruit) et 6 ont pour but de voir si l'algorithme sépare les régions de signal en fonction du signal sans être guidé par le bruit. La configuration 7 a été créée pour voir comment réagit la parcellisation lorsque il y a deux signaux superposés dans une région.

Pour chacune des simulations, nous avons fait varier le SNR et nous avons déterminé le nombre de parcelles à l'aide de l'indice de silhouette simplifiée (SSI) (**cf. équation 6.2**), du coefficient de Calinski-Harabaz (CHI) (**cf. équation 6.4**) et de la *slope statistic* calculée à partir de l'indice de silhouette simplifiée (**cf. équation 6.7**)). Dans les tables 9.6, 9.7 et 9.8, les nombres en gras correspondent au nombre exact de parcelles. Les trois tables mettent en évidence qu'il n'existe aucun indice ayant un taux de réussite de 100%, c'est-à-dire qui parvient à estimer le nombre réel de parcelles, et cela quel que soit le niveau de bruit. Le taux de succès montre également que hormis pour le coefficient de Calinski-Harabaz (CHI), augmenter le ratio signal sur bruit améliore l'estimation du nombre de parcelles (le taux de succès augmente lorsque le SNR augmente).

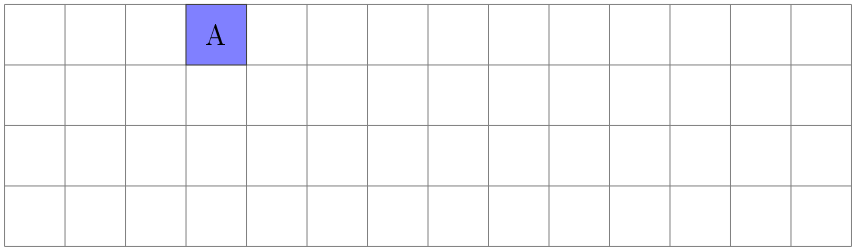
L'indice slope statistic (avec p=0) semble être un indicateur convenable du nombre de parcelles. Cependant, il nécessite de calculer l'indice de silhouette à chaque niveau de parcellisation, autrement dit nV -1 fois, et le coefficient de silhouette pour chaque voxel. Ce calcul est très coûteux. La démarche utilisée a donc été plus exploratoire en étudiant l'évolution des résultats fonctionnels à différentes



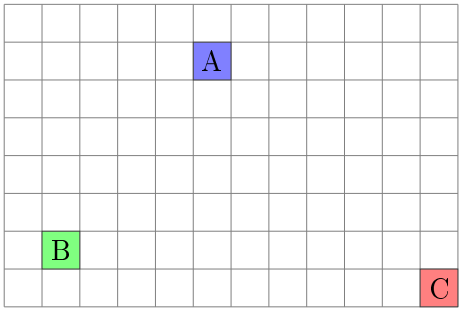
(a) Configuration 1



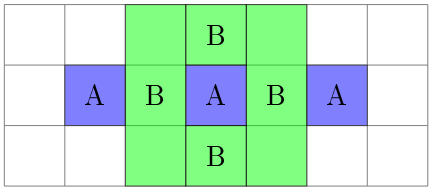
(b) Configuration 2



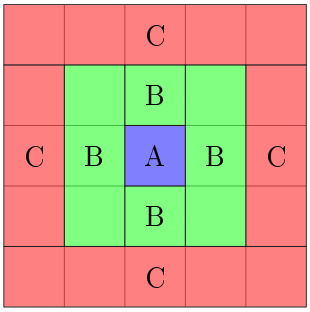
(c) Configuration 3



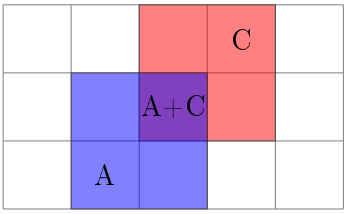
(d) Configuration 4



(e) Configuration 5



(f) Configuration 6



(g) Configuration 7

FIGURE 9.11 – Les différentes configurations des régions.
Les cases blanches représentent des régions de bruit.

échelles.

Simulation	CHI	SSI	slope(p=0)	slope(p=1)	slope(p=2)	slope(p=3)
1	**2**	**2**	**2**	**2**	**2**	**2**
2	2	2	**3**	2	**3**	**3**
3	6	2	**4**	2	**4**	**4**
4	3	2	**4**	**4**	**4**	**4**
5	8	**2**	**2**	**2**	**2**	**2**
6	5	**3**	**3**	**2**	**3**	**3**
7	2	3	**4**	2	**4**	**4**
8	**2**	**2**	**2**	**2**	**2**	**2**
9	3	2	**3**	2	2	2
10	7	**2**	**2**	**2**	**2**	**2**
11	5	2	4	2	2	2
12	5	**2**	10	**2**	10	10
13	5	**3**	**3**	2	**3**	**3**
14	8	2	**3**	2	**3**	**3**
15	7	**2**	**2**	**2**	**2**	**2**
16	6	2	2	2	2	2
17	7	2	**4**	2	**4**	3
18	3	**2**	6	**2**	6	6
19	5	**2**	7	**2**	7	7
Taux de succès	10,53%	52,63%	73,68%	52,63%	68,43%	63,16%

TABLE 9.6 – Nombre de parcelles optimal pour $snr = 200$.

Simulation	CHI	SSI	slope(p=0)	slope(p=1)	slope(p=2)	slope(p=3)
1	3	**2**	**2**	**2**	**2**	**2**
2	8	2	**3**	2	3	3
3	11	2	5	2	2	2
4	5	2	5	5	2	2
5	3	**2**	**2**	**2**	**2**	**2**
6	10	2	**3**	2	**3**	**3**
7	3	2	3	2	3	3
8	5	**2**	3	**2**	**2**	**2**
9	2	2	**3**	2	2	2
10	6	**2**	**2**	**2**	**2**	**2**
11	5	2	**3**	2	2	2
12	3	**2**	6	**2**	6	6
13	4	2	**3**	2	**3**	**3**
14	2	2	**3**	2	**3**	**3**
15	8	**2**	**2**	**2**	**2**	**2**
16	**4**	2	2	2	2	2
17	2	2	**4**	2	3	3
18	4	**2**	10	**2**	10	10
19	6	**2**	7	**2**	7	7
Taux de succès	5,26%	42,11%	57,49%	42,11%	42,11%	42,11%

TABLE 9.7 – Nombre de parcelles optimal pour $snr = 100$.

Bien que le nombre de parcelles estimé ne corresponde pas à la réalité, lorsqu'on découpe les données en spécifiant le nombre correct de parcelles, on constate que hormis pour les simulations n^{os} 3 et 4, on retrouve le découpage initial en région de bruit et de signal. En particulier, pour la simulation n^o 16, pour quatre parcelles, on retrouve le découpage en quatre régions. Pour la simulation n^o 3 (**cf. Figure 9.12(a)**), certains voxels sont associés à la mauvaise parcelle, bien que dans l'ensemble on retrouve la configuration choisie. Cela semble dû au bruit car le résultat diffère suivant le SNR choisi : le nombre de voxels mal classés diminue lorsque le SNR

Simulation	CHI	SSI	slope(p=0)	slope(p=1)	slope(p=2)	slope(p=3)
1	12	**2**	**2**	**2**	**2**	**2**
2	2	**3**	**3**	**2**	**3**	**3**
3	5	**3**	**4**	2	**4**	**4**
4	10	3	**4**	**4**	**4**	**4**
5	11	**2**	**2**	**2**	**2**	**2**
6	3	**3**	**3**	2	**3**	**3**
7	9	3	**4**	2	**4**	**4**
8	7	**2**	3	**2**	**2**	**2**
9	2	2	**3**	2	2	2
10	9	**2**	3	**2**	**2**	**2**
11	7	2	**3**	2	2	2
12	4	**2**	5	**2**	5	5
13	5	**3**	**3**	2	**3**	**3**
14	9	2	**3**	2	**3**	**3**
15	9	**2**	**2**	**2**	**2**	**2**
16	8	2	2	2	2	2
17	5	2	**4**	2	**4**	**4**
18	4	**2**	7	**2**	7	7
19	5	**2**	9	**2**	9	9
Taux de succès	0%	63,16%	68,42%	52,63%	68,42%	68,42%

TABLE 9.8 – Nombre de parcelles optimal pour $snr = 300$.

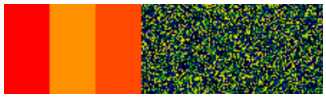
augmente. On retrouve également pour la simulation 1 avec un SNR de 100, quelques voxels mal classés. Comme on ne dispose pas d'indice pour estimer le nombre optimal de parcelles, nous avons regardé sur les simulations ce que donne la surparcellisation c'est-à-dire choisir un nombre de parcelles supérieur au nombre de parcelles optimal. Pour cela, on a choisi un nombre de parcelles égal à 10. Pour la configuration 1 et pour les simulations avec cinq blocs, la région de bruit est parcellisée (**cf. Figure 9.13(b)**). Pour les simulations n^{os} 8 et 10, les deux régions de signal sont découpées en plusieurs régions (**cf. Figure 9.14(b)**). Pour les simulations n^{os} 9 et 11, suivant le SNR, en plus de la région de bruit, les régions de signal peuvent être aussi parcellisée (**cf. Figure 9.16(b)**). Pour la configuration 2, la région de signal est parcellisée en plusieurs parcelles qui se répartissent dans les trois blocs de signal (**cf. Figure 9.17(b)**). En outre, suivant le signal sur bruit, la région entourant les trois blocs est également parcellisée. Pour la configuration 4, la région de bruit est divisée en plusieurs régions et la région de signal est intacte (**cf. Figure 9.18(b)**). Pour la configuration 5, la région contenant le signal B est parcellisée en sous-régions alors que la région de signal A et la région de bruit sont intactes (**cf. Figure 9.15(b)**). Pour la configuration 6, la couronne extérieure est divisée en plusieurs parcelles. Enfin, pour la configuration 7, la région de bruit et la région de superposition des deux signaux restent intactes et ce sont les régions contenant les signaux A et B qui sont divisées (**cf. Figure 9.19(b)**). Concernant la configuration 7, lorsqu'on divise l'image en deux parcelles, on constate que les régions contenant seulement le signal A et le signal B sont groupées avec la région de bruit (**cf. Figure 9.19(c-d)**). **L'effet de la surparcellisation dépend donc de la configuration spatiale des régions et aussi du SNR.**

(a) Simulation 3- Nombre optimal de parcelles

(b) Simulation 3- $n_C = 10$

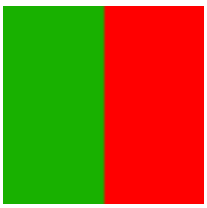
FIGURE 9.12 – Effet de la surparcellisation pour la simulation 3 .

(a) Simulation 4- Nombre optimal de parcelles

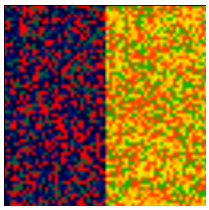


(b) Simulation 4- $n_C = 10$

FIGURE 9.13 – Effet de la surparcellisation pour la configuration 1 avec cinq blocs (1 couleur par parcelle).

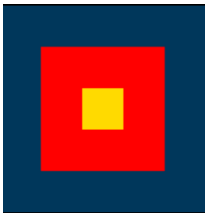


(a) Simulation 8- Nombre optimal de parcelles

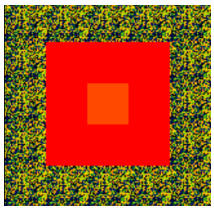


(b) Simulation 8- $n_C = 10$

FIGURE 9.14 – Effet de la surparcellisation pour la configuration 1 sans zone de bruit (1 couleur par parcelle) .



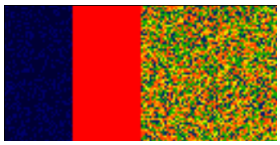
(a) Simulation 14- Nombre optimal de parcelles



(b) Simulation 14- $n_C = 10$

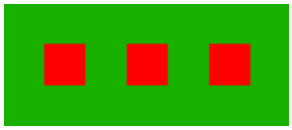
FIGURE 9.15 – Effet de la surparcellisation pour la configuration 6 (1 couleur par parcelle) .

(a) Simulation 11- Nombre optimal de parcelles

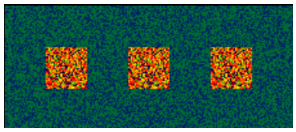


(b) Simulation 11- $n_C = 10$

FIGURE 9.16 – Effet de la surparcellisation pour la configuration 1 autant de région de signal que de région de bruit (1 couleur par parcelle) .



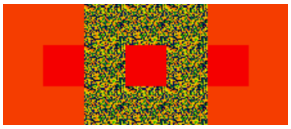
(a) Simulation 12- Nombre optimal de parcelles



(b) Simulation 12- $n_C = 10$

FIGURE 9.17 – Effet de la surparcellisation pour la configuration 2 (1 couleur par parcelle).

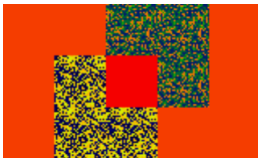
(a) Simulation 13- Nombre optimal de parcelles



(b) Simulation 13- $n_C = 10$

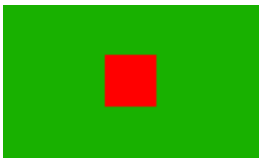
FIGURE 9.18 – Effet de la surparcellisation pour la configuration 5 (1 couleur par parcelle).

(a) Simulation 16- Nombre optimal de parcelles



(b) Simulation 16- $n_C = 10$

(c) Simulation 16- $n_C = 3$



(d) Simulation 16- $n_C = 2$

FIGURE 9.19 – Évolution du nombre de parcelles pour la configuration 7 (1 couleur par parcelle).

Les simulations confirment les bonnes performances de l'algorithme de Ward pour agglomérer les séquences fonctionnelles en parcelles, en restant totalement guidé par les données. Elles soulignent aussi l'instabilité des résultats lorsque le nombre de parcelles demandé est supérieur au vrai nombre de parcelles (surparcellisation). C'est pourquoi les résultats des parcellisations seront analysés d'abord avec un faible nombre de parcelles (100).

9.3 Parcellisation des données à différentes échelles

9.3.1 Quelques cartes de parcellisations

Nous testons la parcellisation de groupe par concaténation des données et la parcellisation individuelle pour chaque sujet. Cependant, pour concaténer les signaux de chaque sujet, on ne garde que les voxels présentant du signal (en éliminant les voxels avec un signal très faible) et communs à tous les sujets. C'est pour cela que le nombre de voxels parcellisés est moindre pour la parcellisation de groupe (de l'ordre de 160 000) que pour les parcellisations individuelles (de l'ordre de 230 000). La figure 9.20 montre la coupe axiale z=0 mm MNI pour la parcellisation de groupe (à droite) et pour chacune des parcellisations individuelles avec une censure à 95% ; chaque couleur représente une parcelle différente. Sur la première ligne de la figure 9.20, on s'aperçoit que la parcellisation en 100 parcelles change d'un sujet à l'autre. Par exemple, pour les sujets n^{os} 2 et 4, la partie frontale (partie en haut de l'image **cf. Section A.2**) est plus morcelée que pour les sujets n^{os} 1 et 3. À l'inverse, la partie occipitale (en bas de l'image **cf. Section A.2**) est regroupée dans une parcelle pour le sujet n^{o} 4 alors qu'elle est recouverte par quelques parcelles pour les sujets n^{os} 1 et 2 et qu'elle est très morcelée pour le sujet n^{o} 3. Les noyaux

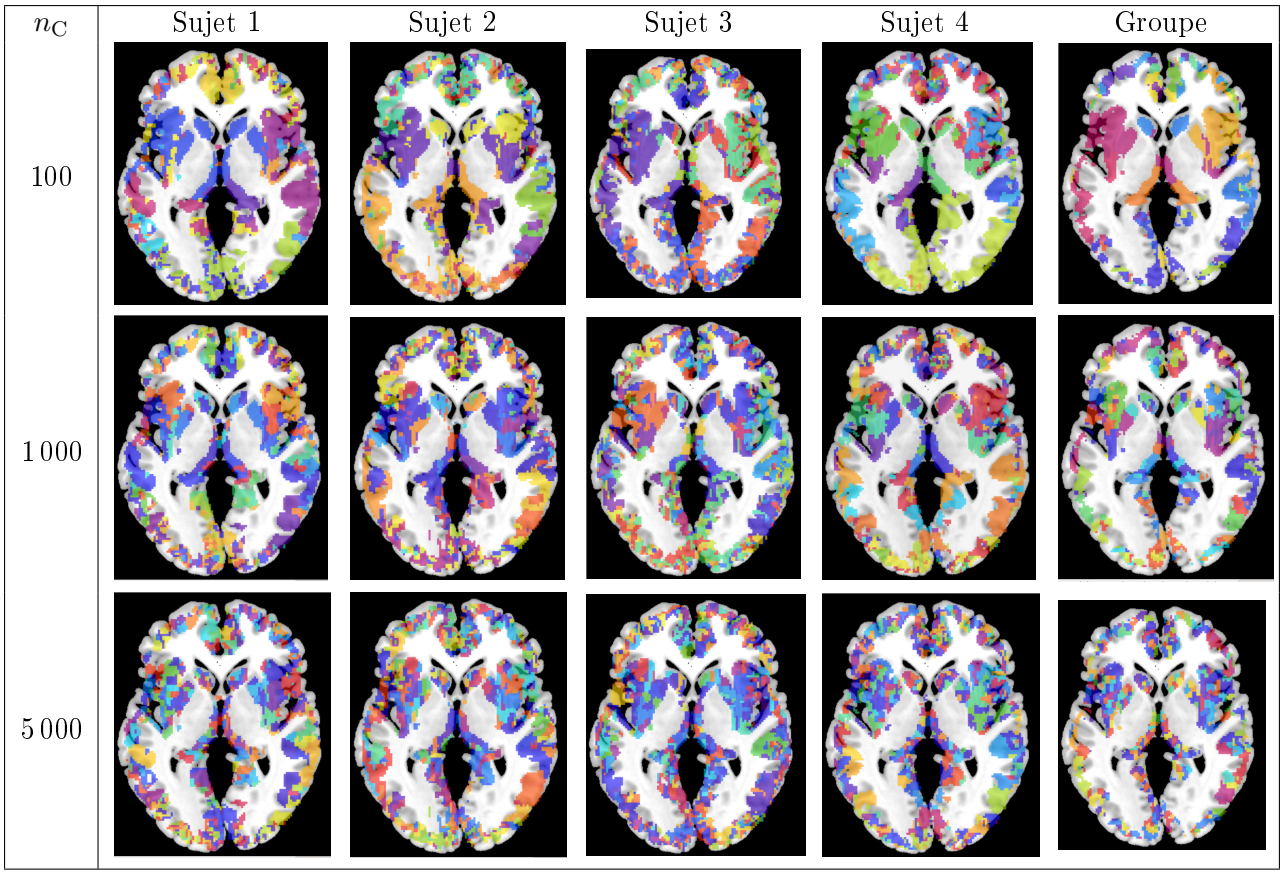


FIGURE 9.20 – Parcellisation individuelle et de groupe à différentes échelles.
Chaque couleur représente une parcelle.

caudés (**cf. Section A.2**) sont dans une seule parcelle pour le sujet n^o 1 alors que pour les autres sujets, certains voxels appartiennent à d'autres parcelles. De part et d'autre des noyaux caudés, on observe pour les parcellisations individuelles une dissymétrie gauche-droite : la partie droite semblant plus morcelée que la partie gauche. Pour la parcellisation de groupe, les noyaux caudés sont dans une parcelle, les parties frontale et orbitale (**cf. Section A.2**) sont moins morcelées que pour les parcellisations individuelles et on observe une plus grande symétrie entre la gauche et la droite. La deuxième ligne de la figure 9.20 montre la parcellisation en 1 000 parcelles. On constate de nouveau que les parcellisations individuelles sont très variables : la partie frontale est plus morcelée pour les sujets 2 et 4 que pour le sujet 1. Pour le sujet 1, la partie à gauche des noyaux caudés est recouverte de plus de parcelles que la partie droite, alors que pour les autres sujets, la partie droite est plus morcelée. La parcellisation de groupe présente un compromis entre les différents sujets. Pour les parcellisations en 5 000 parcelles (Figure 9.20 $3^{ème}$ ligne), les constats sont les mêmes : les parcellisations individuelles sont assez dissemblables et la parcellisation de groupe semble être un compromis entre les différentes parcellisations individuelles.

Si l'inspection visuelle des parcellisations est délicate, en particulier pour les petites échelles, il faut souligner que sans aucune contrainte spatiale les parcelles ne sont pas éparses et semblent correspondre à des territoires neuro-anatomiques. De plus, la parcellisation obtenue est relativement symétrique. Ces éléments sont encourageants et doivent être affinés au moyen de critères adaptés.

9.3.2 Comment évaluer la parcellisation ?

Puisqu'une parcelle doit regrouper des voxels ayant un signal similaire, plusieurs critères fonctionnels ont été définis. Notons $s^i(v,t)$ le signal dans le voxel v au temps t appartenant à

la parcelle i. On peut mesurer la variance intra-parcelle du signal au temps t dans la parcelle i $var^i(t) = var(\{s(v,t), v \in C_i\})$. Cette variance peut être moyennée sur le temps et sur l'ensemble des parcelles en un seul critère. On obtient donc un critère de variance intra-sujet qui donne une indication sur l'homogénéité du signal dans les parcelles. Sous cette forme, le critère est absolu. Pour comparer la taille de l'effet, la variance a été rapportée à la ligne de base calculée dans la parcelle (moyenne selon v) : $var^i(t) = var(\{s(v,t) - \overline{s^i(t)}, v \in C_i\})$ où $\overline{s^i(t)}$ représente le signal moyen dans la parcelle i au temps t. Un autre critère est de mesurer le résidu de signal $res(v,t) = s(v,t) - \overline{s^i(t)}$ et le moyenner sur le temps. Un critère pertinent est de chercher si de l'information est présente dans le signal moyen de la parcelle. On doit rechercher un « lien » avec le paradigme, sachant que de manière générale il est constitué d'un enchaînement non périodique de stimuli de plusieurs types. L'idée la plus simple est de calculer la corrélation moyenne entre tous les signaux issus du même type de stimulation (par exemple les phases de réception de la boisson sucrée). Pour n'obtenir qu'un seul critère, le critère de corrélation est moyenné sur l'ensemble des stimulations considérées puis sur l'ensemble des parcelles. On considère ici les deux phases de réception de la boisson (sucre ou eau) et on moyenne la valeur absolue des deux corrélations en utilisant la transformation de Fisher. Pour cela, les deux coefficients de corrélation sont convertis en z de Fisher ; la moyenne des deux z de Fisher ainsi obtenus est transformée en coefficient de corrélation par la transformée inverse de Fisher. Cette procédure est nécessaire car le coefficient de corrélation n'est pas une mesure linéaire de la force [42]. En passant par la transformation de Fisher, les corrélations sont moins atténuées. L'utilisation de la corrélation a toutefois un inconvénient majeur : elle nécessite un modèle, ce qui est contraire au principe de la méthode pour déterminer la durée de l'onset. De plus, il faudrait l'associer au GLM pour tenir compte du délai de la réponse hémodynamique (ce qui n'a pas été fait ici).

Pour s'affranchir du paradigme, une idée a été de construire des cartes de corrélations entre les parcelles. Pour cela, on calcule le signal moyen dans la parcelle puis pour chaque paire de parcelles, on corrèle les deux signaux moyens. On obtient ainsi une matrice de corrélations avec des 1 sur la diagonale (puisqu'on corrèle le même signal). Cette matrice de corrélations peut être ensuite représentée par carte de chaleur où chaque couleur représente un coefficient de corrélation. Cette carte permet d'identifier les régions cérébrales ayant un signal similaire.

9.3.3 Étude des paramètres fonctionnels en fonction du nombre de parcelles

La figure 9.21 représente pour chaque sujet la boîte à moustache de la corrélation pour chaque sujet avec à gauche la parcellisation individuelle et à droite la parcellisation de groupe. On constate que pour tous les sujets, la corrélation médiane est presque nulle et est stable en fonction du niveau de parcellisation, ce qui signifie que la moitié des parcelles n'est pas corrélée avec la première phase du paradigme. Cette médiane est correcte puisqu'il serait surprenant que toutes les parcelles soient fortement corrélées avec le paradigme. La valeur du maximum (hors données considérées comme aberrantes) augmente un peu pour un découpage en 500 et 1 000 parcelles excepté pour le sujet 2. Le maximum de corrélation est aux alentours de 0,2 donc une corrélation faible. Le sujet 2 a des parcelles avec une corrélation à 0,5, ce qui est assez élevé si on tient compte du bruit dans le signal. L'intervalle entre le premier et le troisième quartile diminue lorsqu'on augmente le nombre de parcelles, mais il se stabilise à partir de 10 000 parcelles. D'une part, à une échelle grossière (un nombre de parcelles petit) les parcelles sont plus grandes donc le bruit présent dans le signal est diminué, ce qui peut expliquer pourquoi la corrélation est plus élevée. De plus à une échelle grossière, les parcelles ont plus de chances de contenir des voxels contenant du signal corrélé avec le paradigme. On observe en revanche qu'il y a peu de différences entre la parcellisation de groupe et les parcellisations individuelles.

La figure 9.22 montre l'évolution de la variance en fonction du nombre de parcelles. On remarque là encore que la variance diminue lorsqu'on augmente le nombre de parcelles pour la même raison que le résidu moyen puisqu'à une échelle fine, les parcelles contiennent moins de voxels. Comme pour le résidu moyen, on observe des parcelles avec une très forte variance. On

retrouve également que la variance est plus faible avec la parcellisation de groupe, en partie car le nombre de voxels parcellisés est moindre. Mais cela suggère aussi que la parcellisation de groupe donne une parcellisation plus homogène.

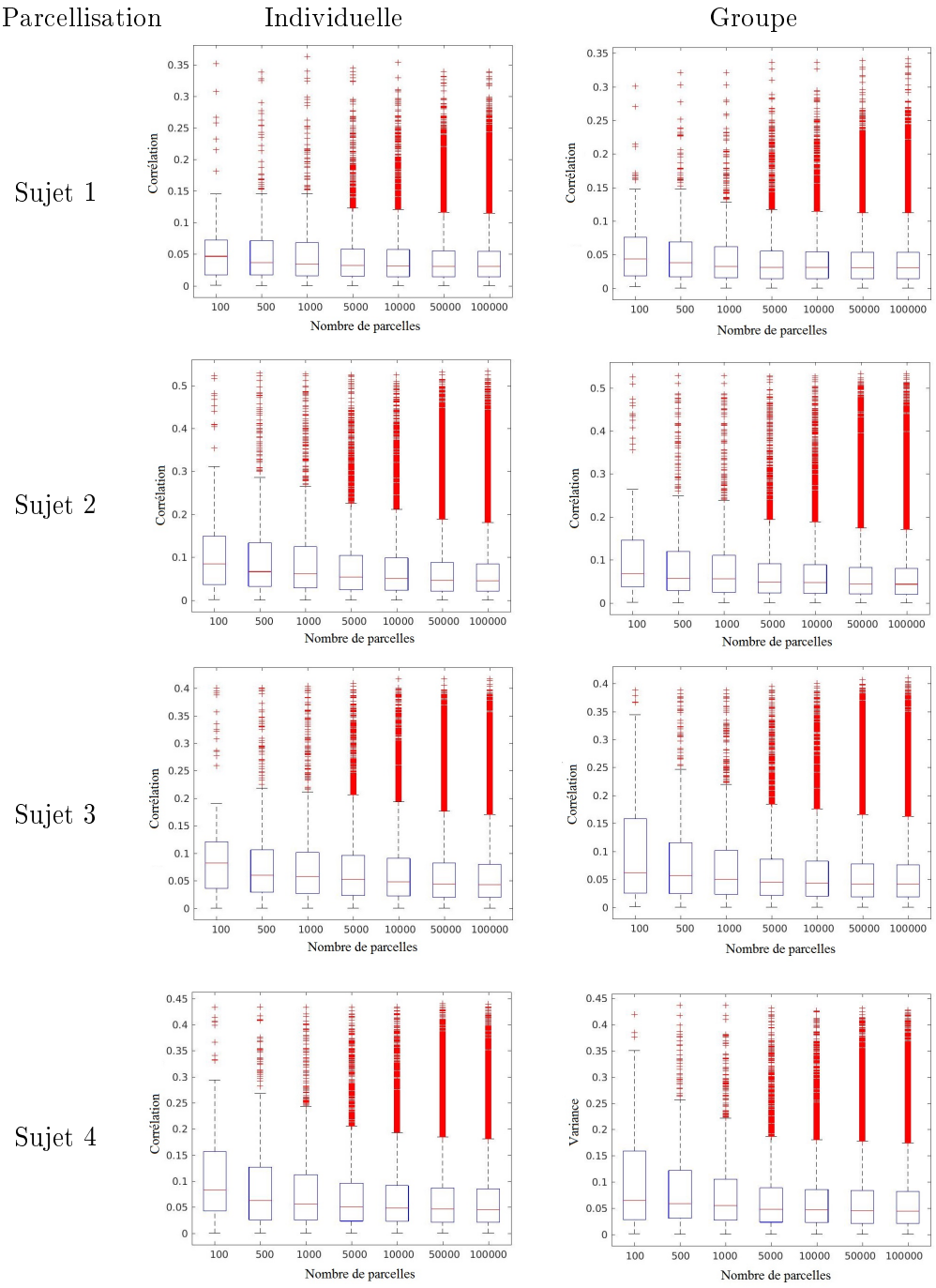


FIGURE 9.21 – Évolution de la corrélation avec le nombre de parcelles.

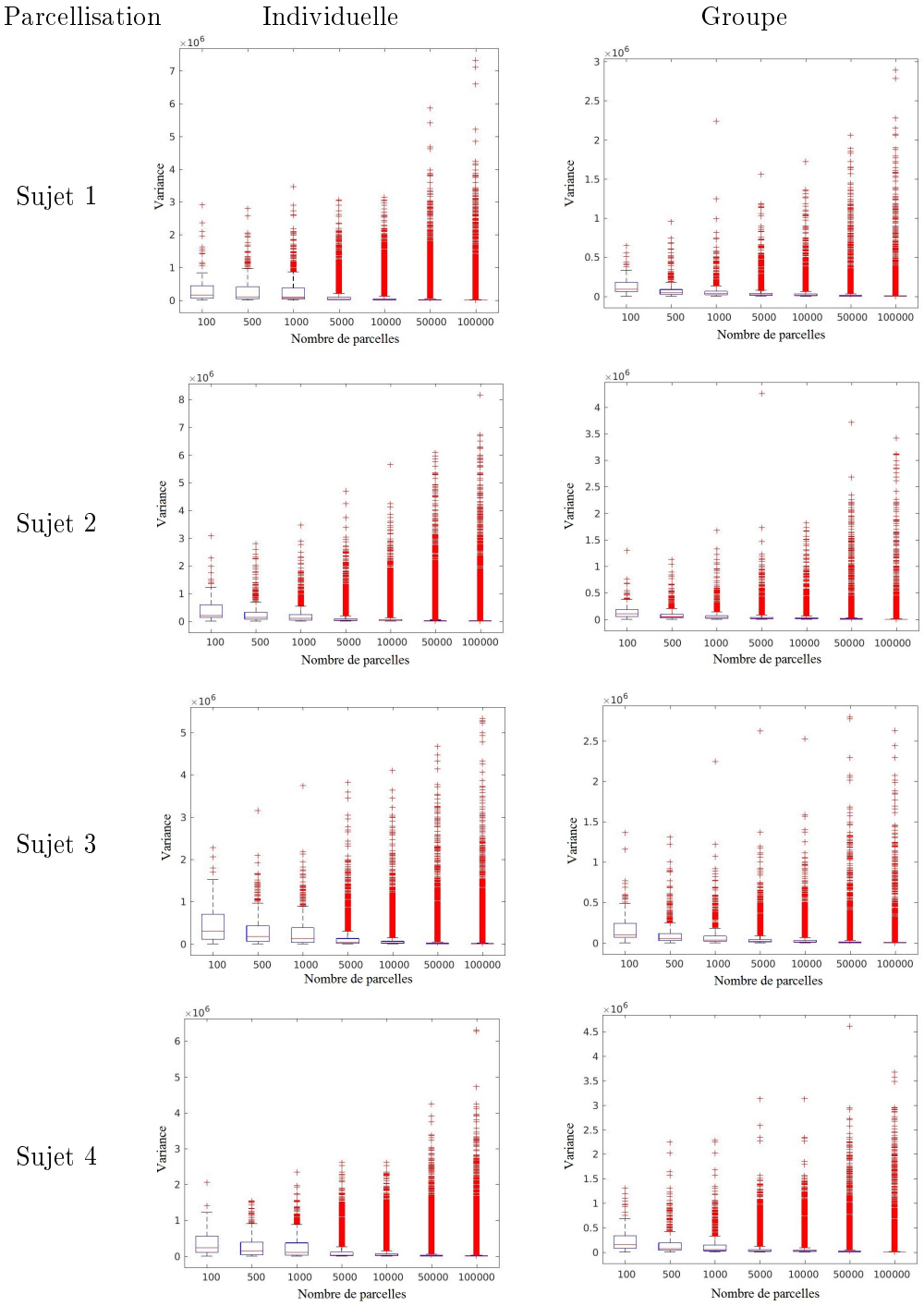


FIGURE 9.22 – Évolution de la variance avec le nombre de parcelles.

9.3.4 Cartes de corrélations entre les parcelles

Calculer la corrélation entre les parcelles permet d'étudier la connectivité des régions cérébrales c'est-à-dire de mettre en lumière les régions cérébrales répondant de manière similaire. L'avantage est d'utiliser l'information fonctionnelle mise en valeur par la parcellisation sans nécessiter de modèle, mais en comparant les signaux deux à deux. Les figures 9.23 à 9.27 montrent les matrices de connectivités obtenues pour chaque sujet à partir de leur parcellisation individuelle et la matrice de connectivités moyennée sur les sujets pour la parcellisation de groupe. Pour labelliser la région anatomique à laquelle appartient chaque parcelle, le coefficient de Dice a été calculé entre la parcelle considérée et les régions d'intérêt issues de l'atlas AAL2 (cf. Section A.2). Chaque parcelle a ensuite été associée à la région pour laquelle le coefficient de Dice est maximal. Sur les matrices de corrélation, les parcelles ont ensuite été regroupées par régions d'intérêt et présentées dans l'ordre de l'atlas AAL2. Autrement dit, toutes les parcelles qui sont majoritairement dans l'insula gauche selon le critère de Dice apparaissent sous le label Insula_L.

Ces matrices de corrélations sont variables d'un sujet à l'autre. Le sujet 1 (**cf. Fig 9.23**) présente des corrélations entre une partie du cortex frontal(Frontal_) et le cortex orbito-frontal (OFC,Frontal_Inf_Tri), entre une partie du cortex gustatif (Insula, Thalamus) et le cortex frontal et cingulaire. Chez le sujet 2 (**cf. Fig 9.24**), nous trouvons également une corrélation entre une partie du cortex frontal et le cortex orbito-frontal, entre le cortex gustatif (insula) et les cortex orbito-frontal (OFC), cingulaire (Cingulate_) et frontal ; et également une corrélation négative entre le gyrus post-central (cortex somato-sensoriel) et le cortex cingulaire, l'insula et le cortex orbito-frontal. Le sujet 3 présente également des corrélations entre le cortex orbito-frontal et le cortex frontal, entre le cortex frontal et le cortex gustatif (insula et thalamus) et entre le cortex orbito-frontal et le cortex gustatif (**cf. Fig 9.25**). Pour le sujet 4, nous retrouvons également cette corrélation entre le cortex frontal et le cortex orbito-frontal(Frontal_Inf_Tri),entre le cortex gustatif (insula et thalamus) et le cortex frontal, le cortex gustatif(insula) et le cortex orbito-frontal(Frontal_Inf_Tri) ; mais également une corrélation entre le gryus post-central (cortex somato-sensoriel) et le thalamus et entre le gyrus pré-central (cortex somato-sensoriel) et l'insula. En moyennant les corrélation sur le groupe, nous observons une corrélation entre le cortex frontal et le cortex orbito-frontal(OFC, Frontal_Inf_Tri), entre le cortex frontal et le cortex gustatif (insula, thalamus), entre le cortex orbito-frontal et le cortex gustatif ; et une corrélation entre le gyrus pré-central et le cortex frontal, le cortex gustatif et le cortex orbito-frontal.

Si on augmente le nombre de parcelles, on retrouve des connectivités similaires mais sur un nombre de parcelles réduit. La répartition dans les région d'intérêts (ROIs) issues de l'atlas changent également puisque les parcelles sont plus petites. Par exemple, la figures 9.28 montre la matrice de corrélations pour un découpage en 1 000 parcelles pour le sujet n^o 1. Les matrices de corrélations des autres sujets et celle issue de la parcellisation de groupe sont données en annexe E.3. La parcellisation de groupe présente donc une meilleure connectivité entre les régions par rapport aux parcellisations individuelles.

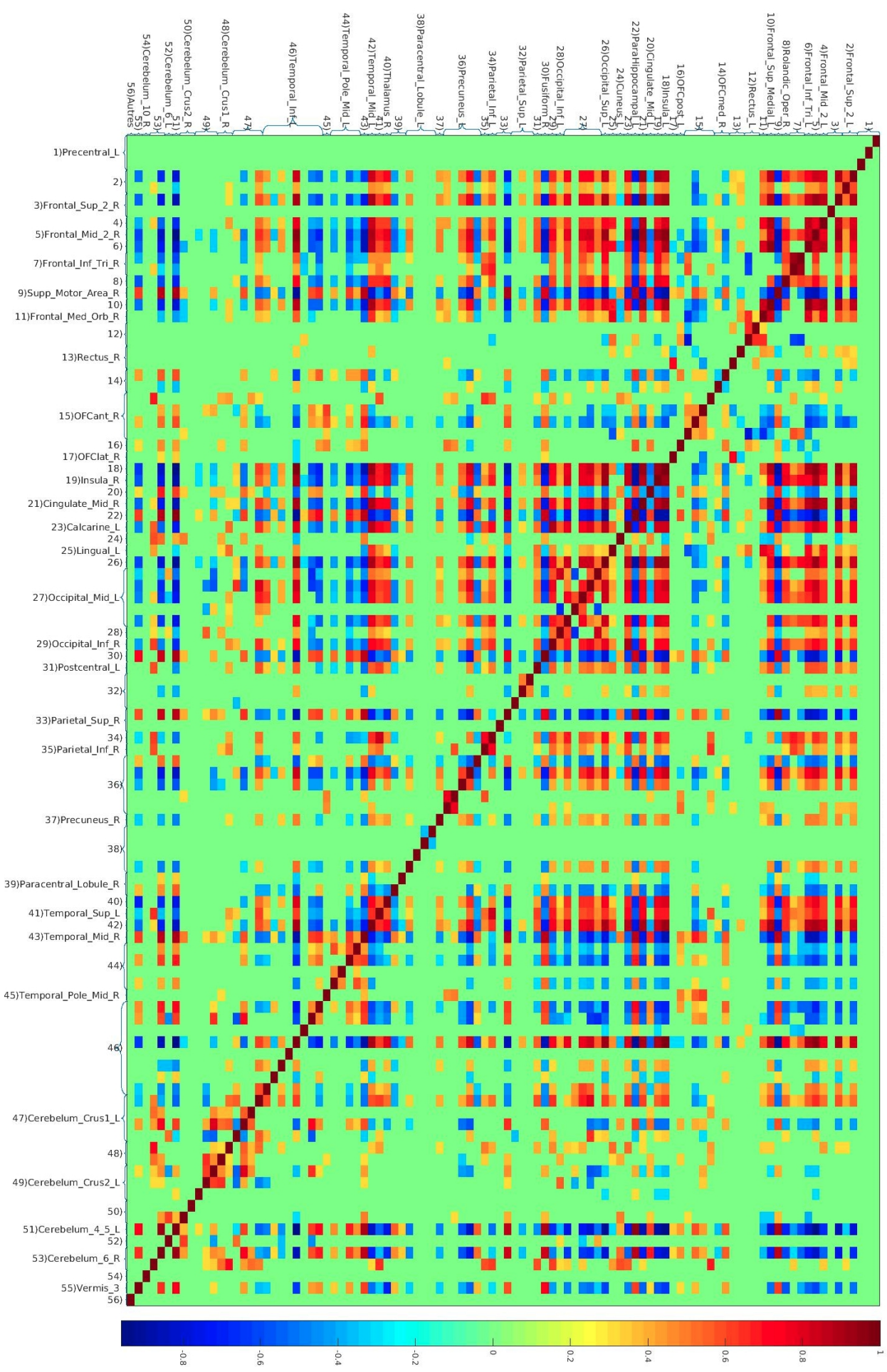


FIGURE 9.23 – Matrice de corrélations entre les 100 parcelles pour le sujet n° 1

Seules les corrélations telles que $|r| \leq 0,3$ sont représentées. Pour aérer les labels des axes, le nom des parcelles avec un numéro pair est donné sur l'axe vertical et le nom des parcelles avec un numéro impair se trouve sur l'axe horizontal. Lorsque son nom n'est pas mentionné, la parcelle est référencée par un numéro.

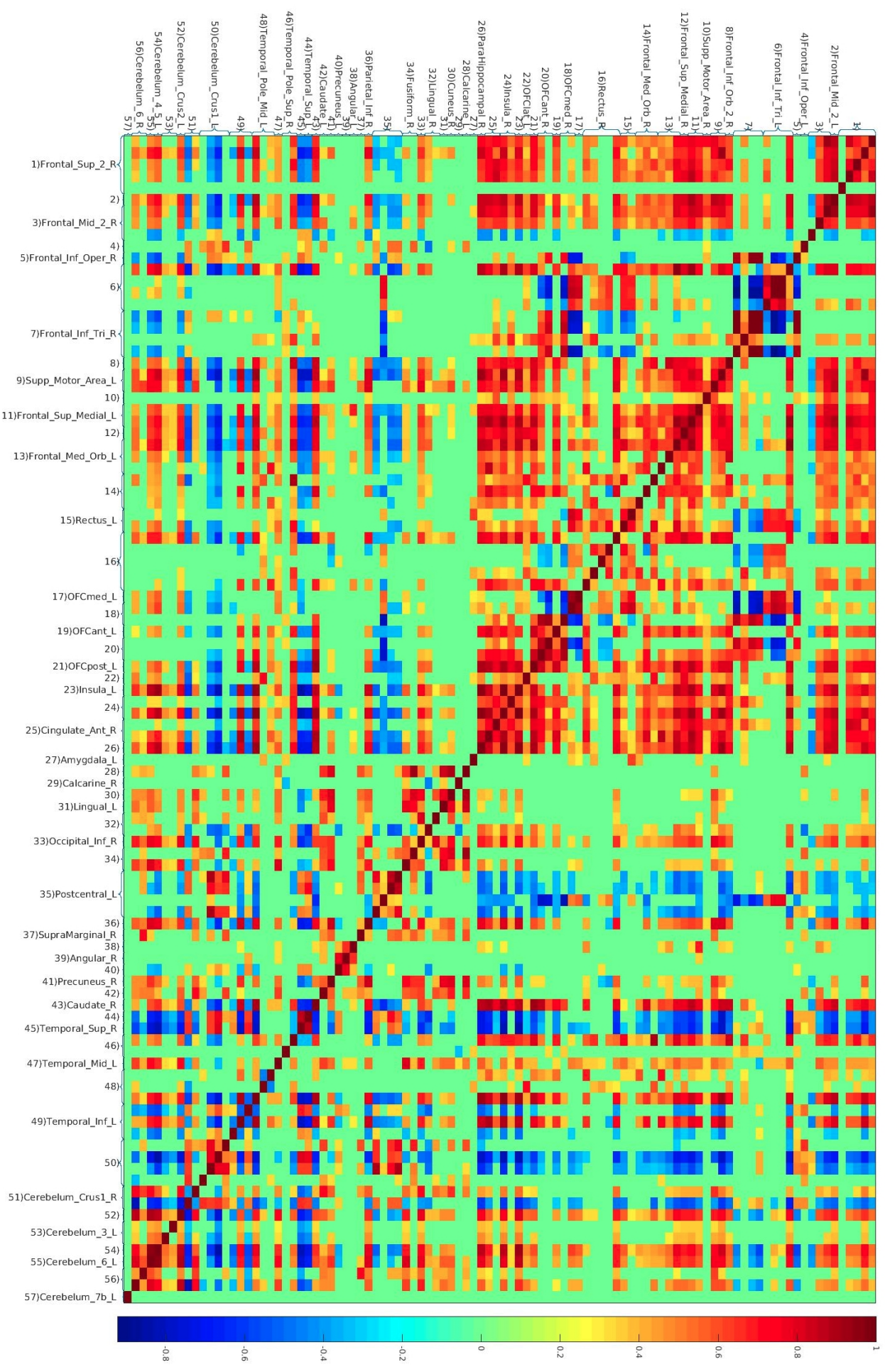


FIGURE 9.24 – Matrice de corrélations entre les 100 parcelles pour le sujet n° 2

Seules les corrélations telles que $|r| \leq 0,3$ sont représentées. Pour aérer les labels des axes, le nom des parcelles avec un numéro pair est donné sur l'axe vertical et le nom des parcelles avec un numéro impair se trouve sur l'axe horizontal. Lorsque son nom n'est pas mentionné, la parcelle est référencée par un numéro.

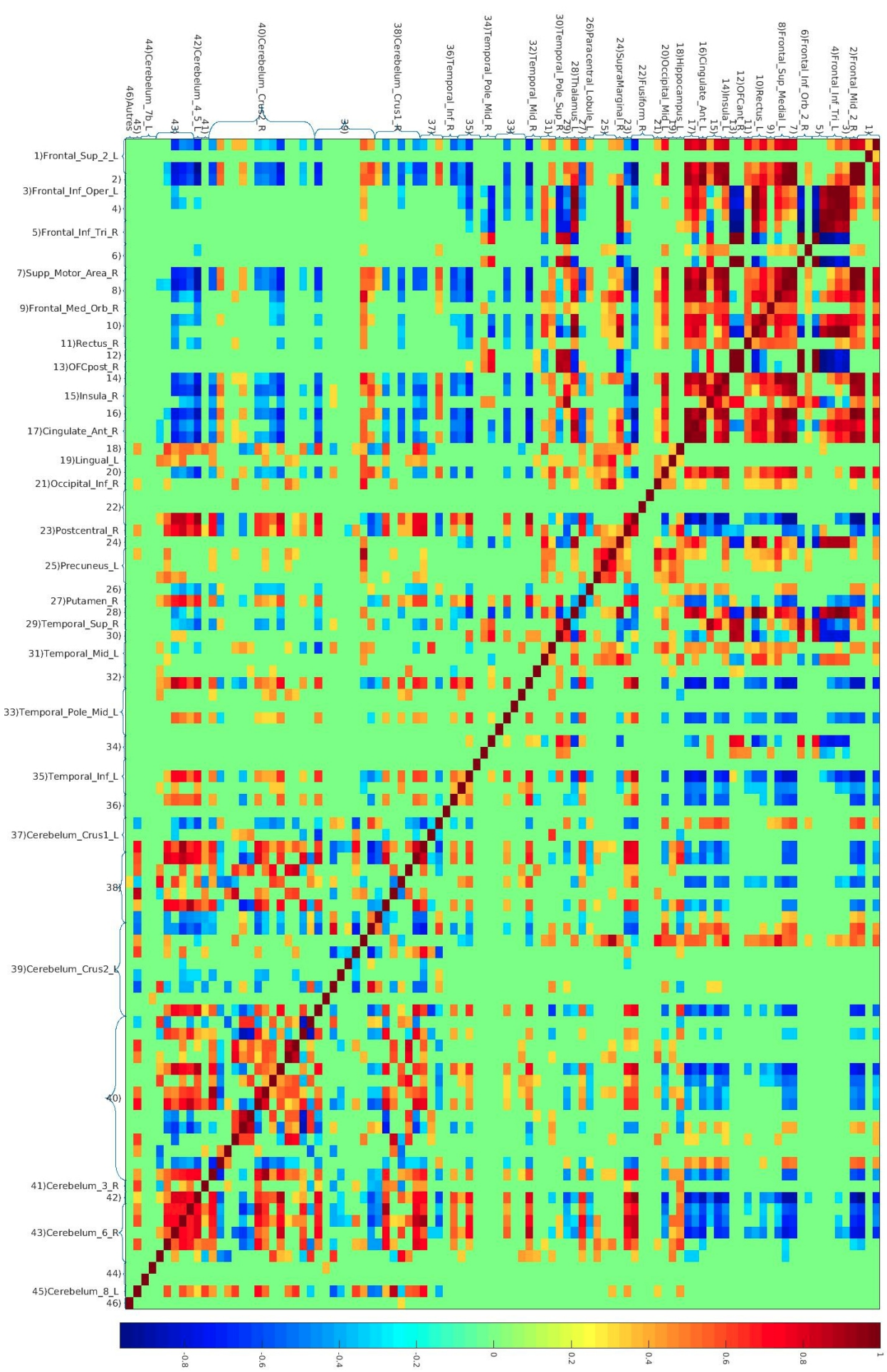


FIGURE 9.25 – Matrice de corrélations entre les 100 parcelles pour le sujet n° 3

Seules les corrélations telles que $|r| \leq 0,3$ sont représentées. Pour aérer les labels des axes, le nom des parcelles avec un numéro pair est donné sur l'axe vertical et le nom des parcelles avec un numéro impair se trouve sur l'axe horizontal. Lorsque son nom n'est pas mentionné, la parcelle est référencée par un numéro.

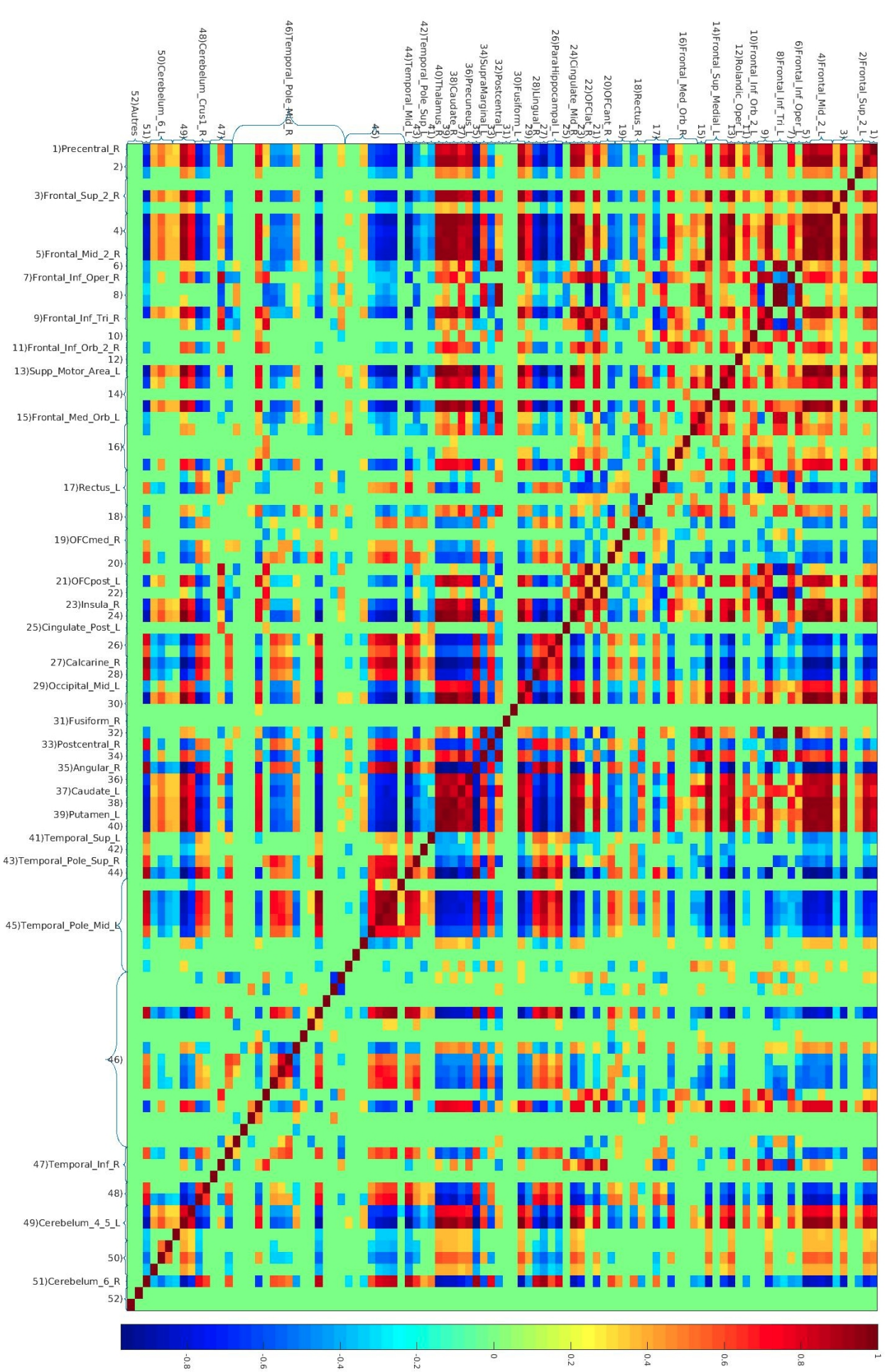


FIGURE 9.26 – Matrice de corrélations entre les 100 parcelles pour le sujet n^o 4

Seules les corrélations telles que $|r| \leq 0,3$ sont représentées. Pour aérer les labels des axes, le nom des parcelles avec un numéro pair est donné sur l'axe vertical et le nom des parcelles avec un numéro impair se trouve sur l'axe horizontal. Lorsque son nom n'est pas mentionné, la parcelle est référencée par un numéro.

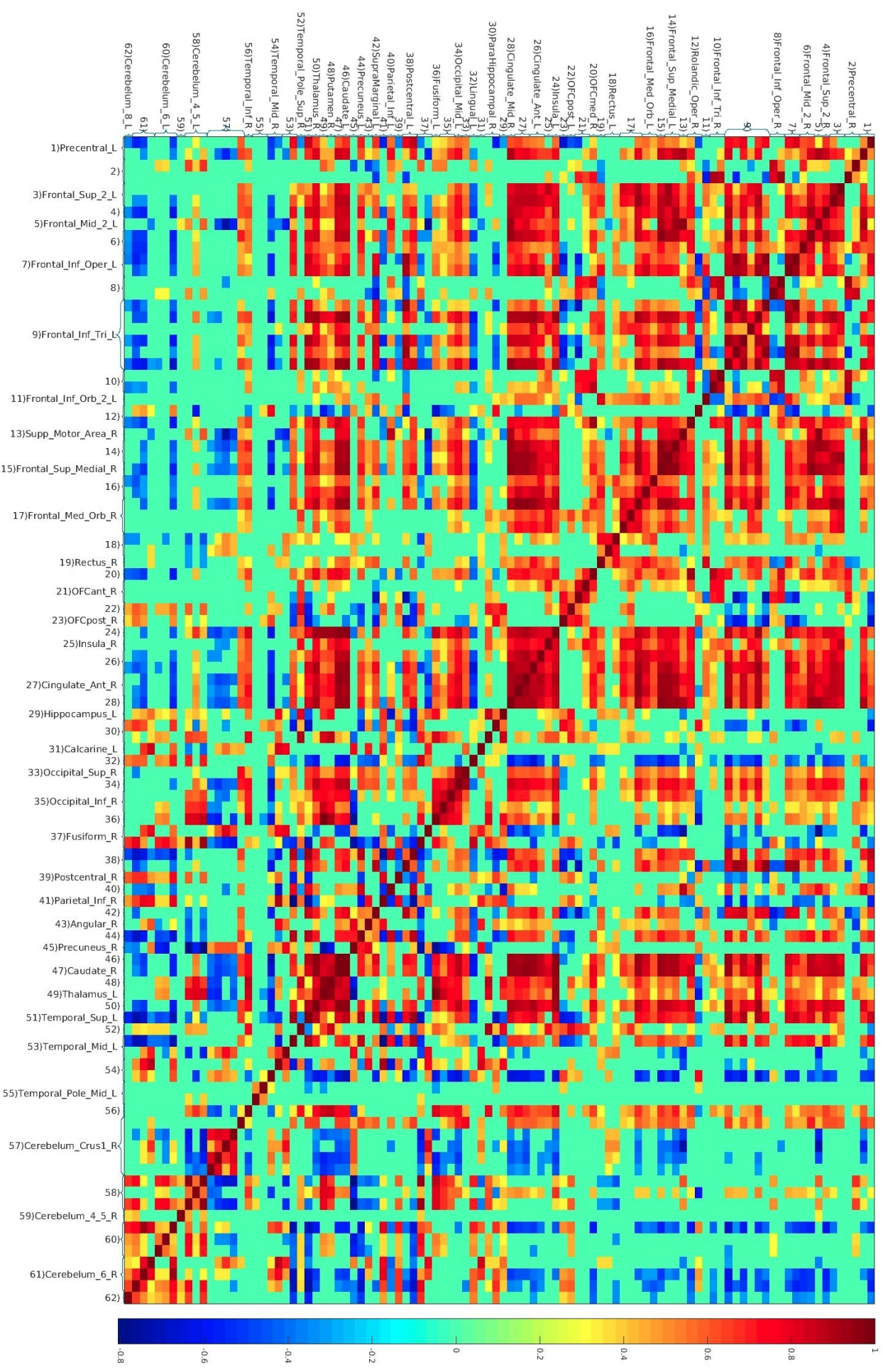


FIGURE 9.27 – Matrice de corrélations entre les 100 parcelles moyennée sur le groupe (parcellisation de groupe)

Seules les corrélations telles que $|r| \leq 0,3$ sont représentées. Pour aérer les labels des axes, le nom des parcelles avec un numéro pair est donné sur l'axe vertical et le nom des parcelles avec un numéro impair se trouve sur l'axe horizontal. Lorsque son nom n'est pas mentionné, la parcelle est référencée par un numéro.

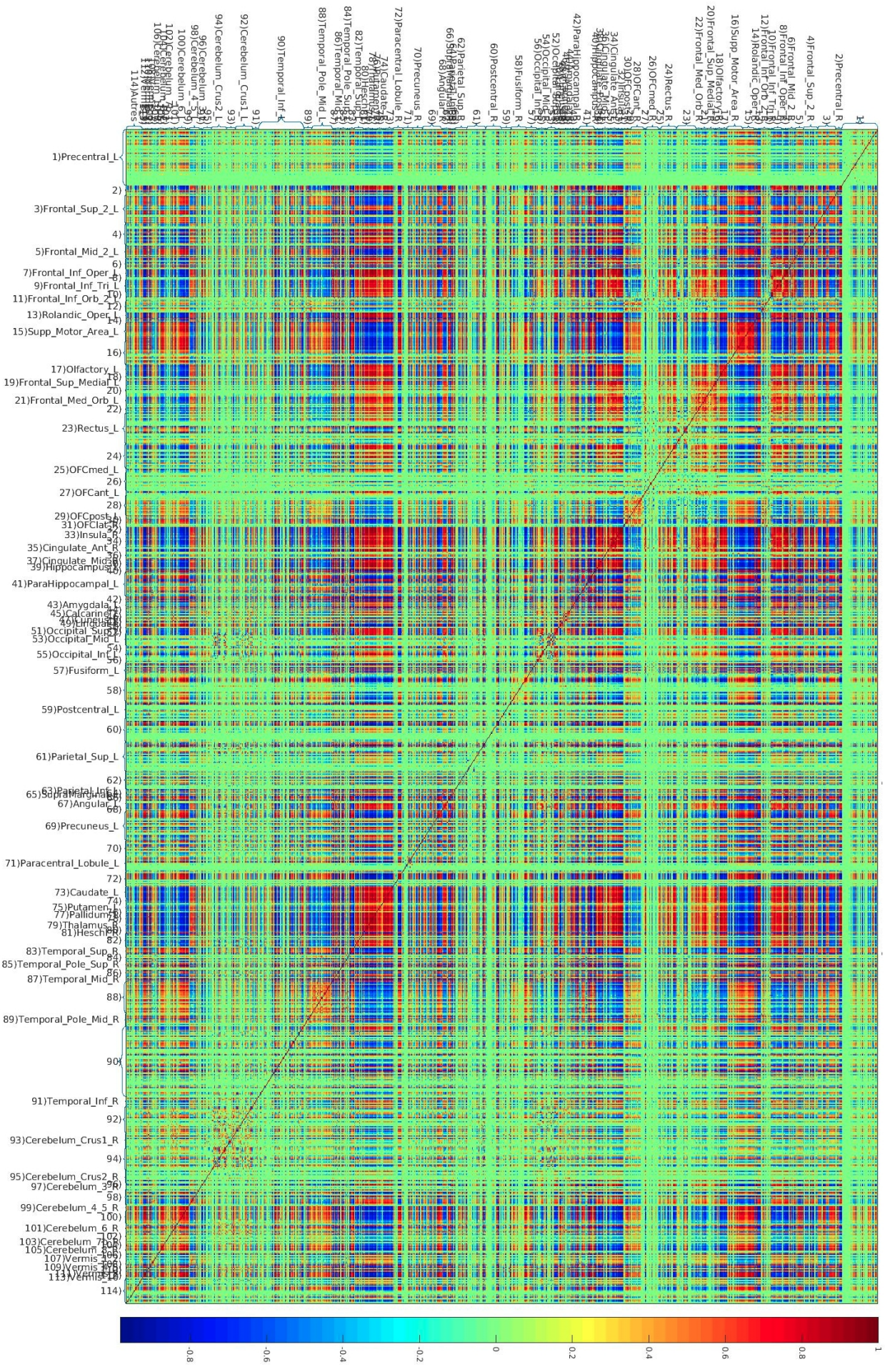


FIGURE 9.28 – Matrice de corrélations entre les 1000 parcelles pour le sujet n^o 1

Seules les corrélations telles que $|r| \leq 0,3$ sont représentées. Pour aérer les labels des axes, le nom des parcelles avec un numéro pair est donné sur l'axe vertical et le nom des parcelles avec un numéro impair se trouve sur l'axe horizontal. Lorsque son nom n'est pas mentionné, la parcelle est référencée par un numéro.

9.3.5 Exemples de signaux trouvés à partir de la corrélation

Nous avons choisi d'explorer les signaux dans les parcelles à un niveau de parcellisation donné. Nous avons pris la parcellisation de groupe afin de disposer de parcelles communes aux sujets et donc d'observer la réponse temporelle dans les mêmes voxels. Nous avons fixé arbitrairement le nombre de parcelles à 5 000 afin d'avoir des parcelles d'une taille équivalente aux activations que donnent habituellement les inférences issues d'un GLM. Nous avons sélectionné les parcelles en fonction de la corrélation avec la phase de réception de boisson du paradigme décrite en section 9.3.2, moyennée sur les sujets (en passant par la transformation de Fisher). Les parcelles les plus corrélées se trouvent dans les régions AAL2 suivantes (**cf. Section A.2**) : le gyrus précentral, le gyrus postcentral, le cortex frontal inférieur operculaire, le cortex frontal inférieur triangulaire et l'operculum rolandique. On trouve également quelques parcelles dans le cortex frontal milieu, le cortex temporal supérieur et milieu, et dans le cortex frontal médial orbital. Les gyrus postcentral fait partie du cortex somato-sensoriel, le gyrus précentral se trouve dans le cortex moteur et le cortex frontal inférieur operculaire appartient quant à lui au cortex gustatif primaire donc il est logique de les trouver lorsqu'on corrèle avec la phase de réception de la boisson. En revanche, les parcelles situées dans l'insula qui fait partie du cortex gustatif sont très faiblement corrélées. Cependant, la corrélation est calculée avec les phases de réception de boisson sans tenir compte du décalage hémodynamique donc il est probable que la réponse temporelle de l'insula survienne pendant une autre phase du paradigme.

Les figures 9.29 à 9.33 montrent la parcelle et le signal dans la parcelle pour chaque sujet. On a choisi des parcelles avec une corrélation moyenne (sur les sujets) supérieure à 0,25 et dans les régions anatomiques probables pour la réception de la boisson. Les signaux montrés ont été échantillonnés toutes les 0,5 secondes en utilisant des splines pour l'interpolation, et ont été moyennés sur les quarante répétitions (ligne noire) et les écarts entre les répétitions sont représentées par les barres d'erreur grises. Les bandes de couleur représentent les différentes phases du paradigme : jaune pour la phase d'attente, rouge pour la phase de réception de la boisson sucrée, bleu pour la phase de réception de l'eau, violet pour la phase d'attente et vert pour la phase de déglutition. La ligne rouge représente le signal moyen calculé sur la phase de repos. Les variations seront décrites ci-après par rapport à cette ligne rouge.

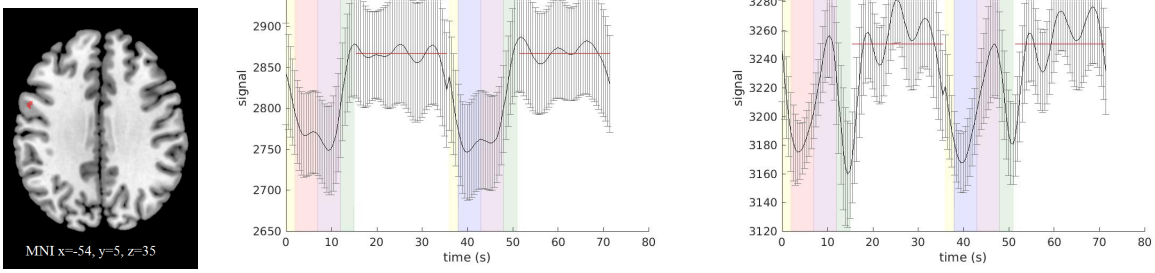
Sur la figure 9.29, on observe le signal dans une parcelle située dans le gyrus pré-central : on constate que pour les sujets n^{os} 1, 3 et 4, il y a une variation négative du signal pendant les phases de réception et d'attente alors que pour le sujet 2, il y a une variation négative suivie d'une variation positive pendant la phase d'attente.

La figure 9.30 montre le signal dans une parcelle située dans à cheval entre le gyrus pré-central et le gyrus post-central : on observe pour les sujets n^{os} 2, 3 et 4 une variation positive pendant les phases de réception et d'attente. En revanche pour le sujet 1, il y a une plus grande variabilité entre les répétitions mais on observe en regardant le signal moyen, une variation positive comme pour les autres sujets.

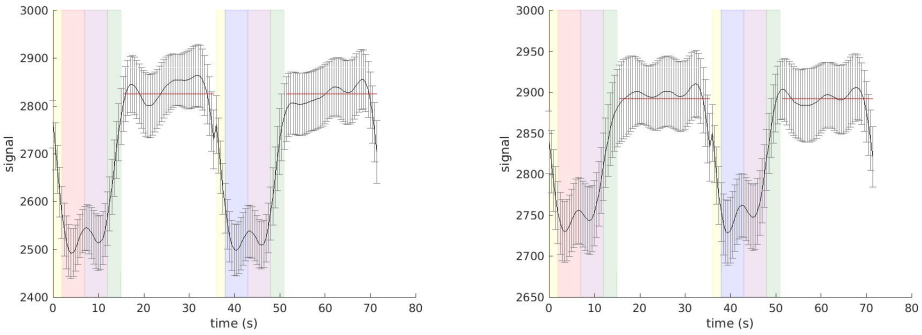
La figure 9.31 correspond à une parcelle dans le gyrus post-central. On observe également une variabilité inter-sujet : pour les sujets 2 et 3, on remarque une variation positive assez marquée pendant la phase de réception et d'attente, pour le sujet 1, il y a également une variation positive mais entre la phase de réception de la boisson sucrée et l'attente et pendant la phase d'attente suivant la phase de réception de l'eau, pour le sujet 4, la variation positive a lieu pendant la phase de réception.

Sur la figure 9.32 on observe la réponse temporelle dans une parcelle située du cortex frontal inférieur operculaire : la réponse des sujets 2 et 3 sont similaires avec une variation positive pendant la phase de réception et d'attente, pour le sujet 4, on retrouve ces variations en plus marquées et pour le sujet 1, la réponse est similaire pour la réception de l'eau mais la variation après la réception de la boisson est plus étendue et forme un pic pendant la phase d'attente.

La figure 9.33 montre la réponse dans une autre parcelle du cortex frontal inférieur operculaire : les réponses des sujets 3 et 4 se ressemblent avec une forte variation positive pendant la

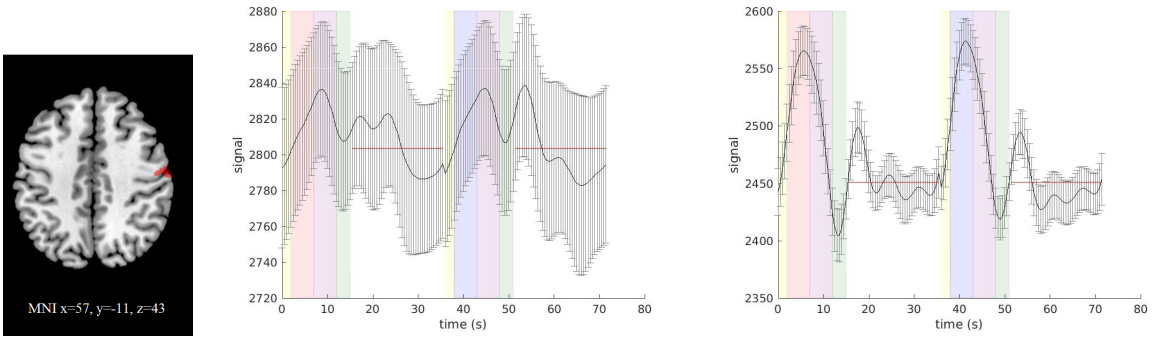


(a) Parcelle (b) Sujet 1 (c) Sujet 2

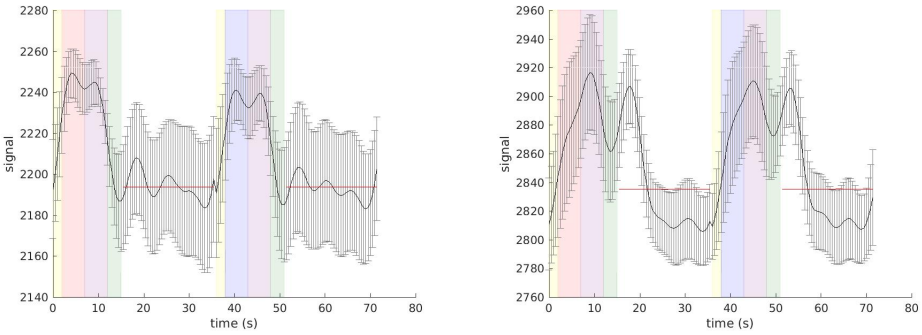


(d) Sujet 3 (e) Sujet 4

FIGURE 9.29 – Signaux dans la parcelle 4755 dans le gyrus pré-central.

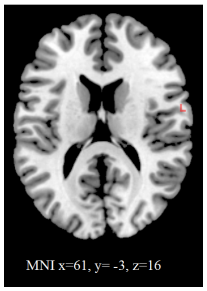


(a) Parcelle (b) Sujet 1 (c) Sujet 2

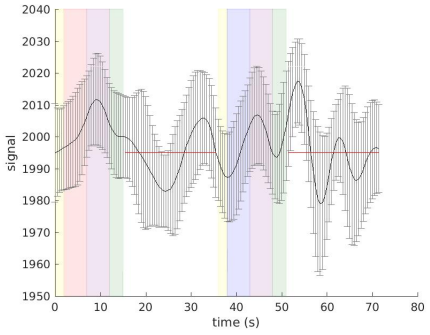


(d) Sujet 3 (e) Sujet 4

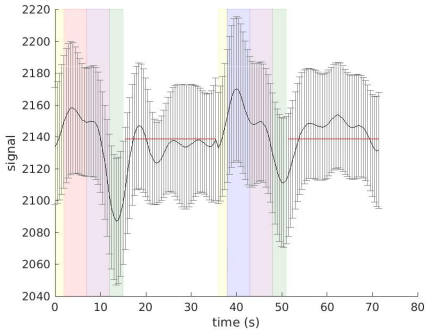
FIGURE 9.30 – Signaux dans la parcelle 3764 dans les gyrus pré-central et post-central .



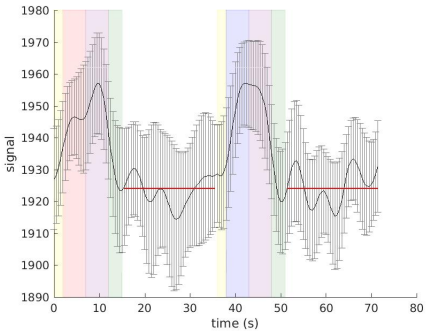
(a) Parcelle



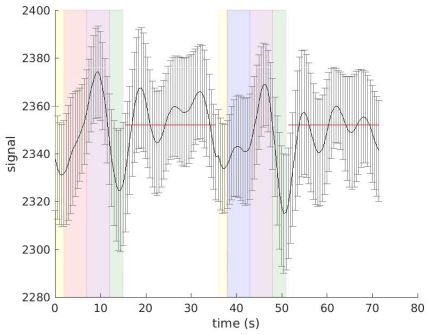
(b) Sujet 1



(c) Sujet 2

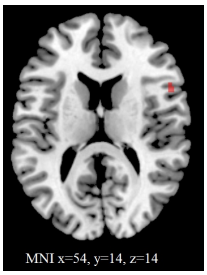


(d) Sujet 3

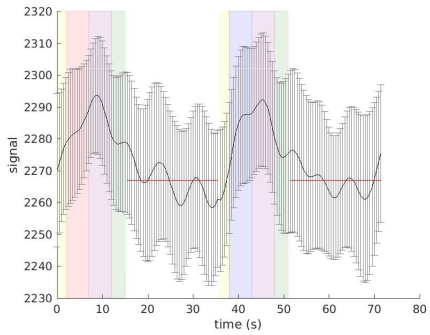


(e) Sujet 4

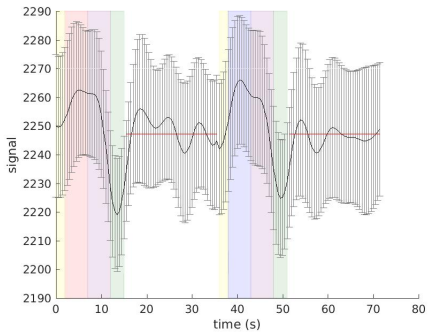
FIGURE 9.31 – Signaux dans la parcelle 4749 dans le gyrus postcentral .



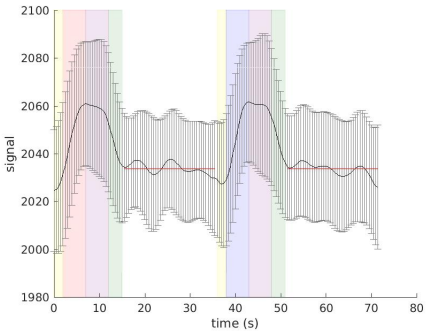
(a) Parcelle



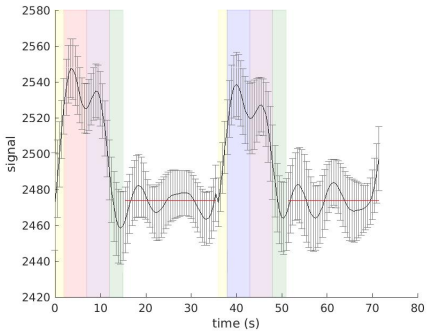
(b) Sujet 1



(c) Sujet 2



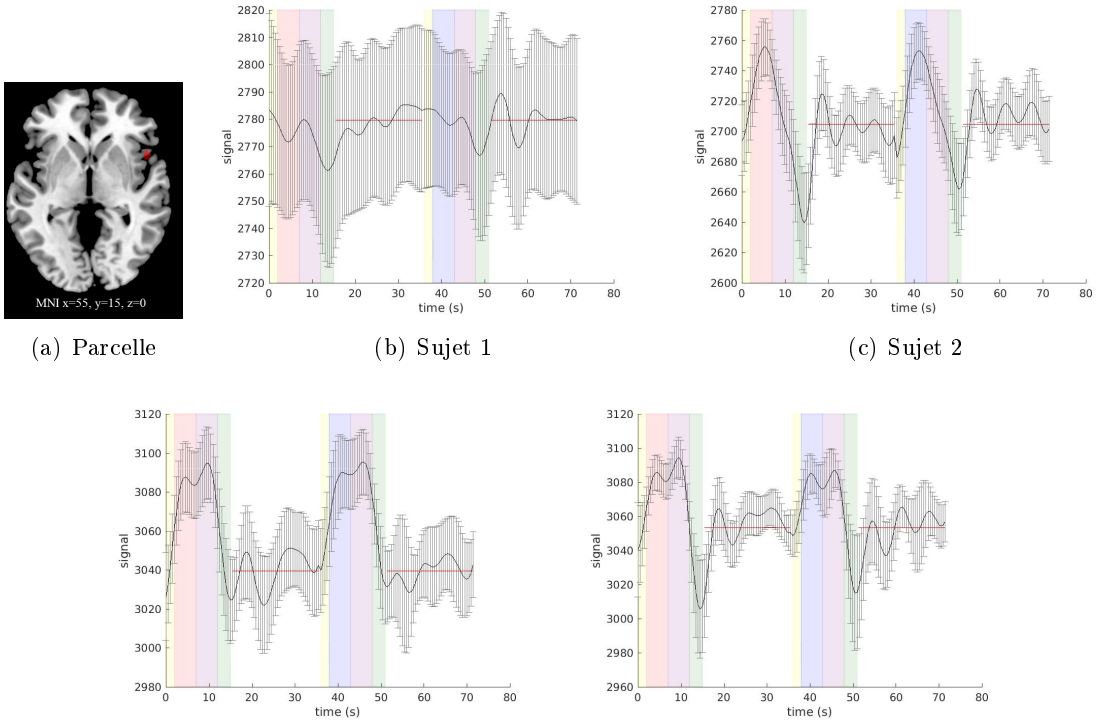
(d) Sujet 3



(e) Sujet 4

FIGURE 9.32 – Signaux dans la parcelle 1200 dans le cortex frontal inférieur operculaire .

phase de réception et d'attente, la réponse du sujet 2 présente une variation positive plus courte dans le temps (qui ne dure que pendant la phase de réception) avec une variation négative pendant la phase de déglutition. En revanche, le sujet 1 ne présente pas de réponse spécifique à la stimulation avec une très forte variation entre les répétitions. Les signaux présentés comportent des variations pendant les phases de réception et d'attente ce qui est normal car la corrélation a été calculée avec la phase de réception.



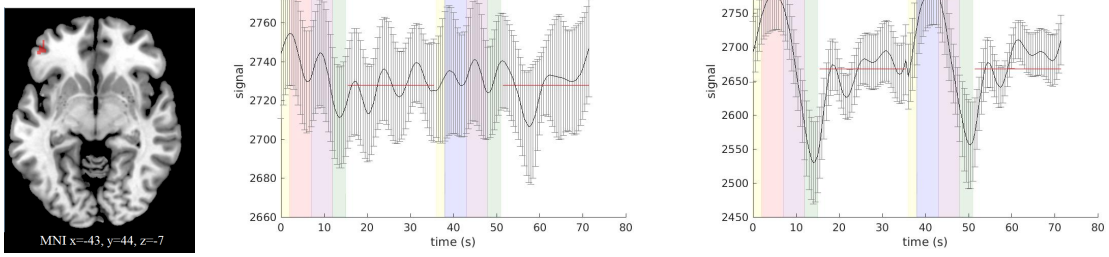
(a) Parcelle (b) Sujet 1 (c) Sujet 2

(d) Sujet 3 (e) Sujet 4

FIGURE 9.33 – Signaux dans la parcelle 1943 dans le cortex frontal inférieur operculaire .

La figure 9.34 montre le signal dans une parcelle du cortex frontal médian : pour les sujets n^{os} 2, 3 et 4 on observe une belle réponse avec une variation positive pendant les phases de réception et d'attente mais pour le sujet 1, la réponse comporte plusieurs pics pendant toutes les phases et ne semble donc pas spécifique à la réception de la boisson.

Nous avons ainsi trouvé des parcelles avec des signaux assez similaires entre les sujets et dans des régions fonctionnelles cohérentes avec le paradigme. En revanche, certaines parcelles n'ont pas de signaux similaires entre les sujets, ce qui montre la limite de la parcellisation de groupe par concaténation qui ne calcule pas de similarité entre les réponses des différents sujets. Toutefois, l'exploration des parcelles bien que fastidieuses permet de faire ressortir des signaux avec une grande variété de formes.



(a) Parcelle (b) Sujet 1 (c) Sujet 2

(d) Sujet 3 (e) Sujet 4

FIGURE 9.34 – Signaux dans la parcelle 1525 dans le cortex frontal médian .

Quatrième partie

Conclusion

Chapitre 10

Conclusion

10.1 Résumé des contributions

L'emploi d'une stimulation alimentaire lors d'une expérience d'IRMf permet d'identifier les régions impliquées dans l'intégration cérébrale de ce signal, depuis celles codant la flaveur jusqu'à des informations cognitives comme la valeur. Cependant, en plus des problèmes inhérents à l'acquisition de telles données, leur analyse est difficile en raison notamment de l'incertitude et des perturbations liées à l'exécution du paradigme et des variabilités intra- et inter-sujet des réponses aux stimuli chimiques. L'enjeu principal de cette thèse était de **proposer une méthode d'analyse adaptée à ces données et guidée par elles seules.** Notre choix s'est porté sur la parcellisation qui consiste à regrouper les voxels ayant une réponse similaire. Plusieurs auteurs ont obtenu des résultats satisfaisants en employant la parcellisation sur des données d'IRMf [202, 160, 81]. Cependant, ils ont tous réduit la dimension des données parcellisées afin d'exécuter la parcellisation en un temps raisonnable. Cette thèse a permis de parcelliser les données d'IRMf sans réduction préalable. Ceci a été rendu possible en développant une implémentation de l'algorithme de Ward à l'aide du calcul intensif.

Nous avons donc optimisé et parallélisé l'algorithme de Ward. Nous avons concentré nos efforts sur la parallélisation du calcul de la distance euclidienne qui représente plus de 90% du temps d'exécution de l'algorithme en séquentiel. L'optimisation du calcul de la distance euclidienne en séquentiel a abouti à un découpage des lignes et des colonnes de la matrice de données exploitant mieux la localité des données. Nous avons ainsi implémenté un algorithme *cache-aware* pour le calcul de la distance avec la possibilité de déterminer les paramètres optimaux. Nous avons ensuite parallélisé le calcul de la distance et produit trois algorithmes parallèles : un algorithme utilisant OpenMP pour les architectures à mémoire partagée, un algorithme avec CUDA pour les GPUs NVIDIA et un algorithme utilisant MPI pour les architectures à mémoire distribuée. Sur une station de travail, l'emploi des GPUs NVIDIA est plus efficace. Mais sur un serveur de calcul disposant d'un grand nombre de cœurs, l'algorithme avec OpenMP surpasse en double précision l'algorithme en CUDA. Une fois le calcul de la distance parallélisé, les itérations regroupant les voxels deux à deux deviennent le facteur limitant de l'algorithme. Nous avons proposé une parallélisation à l'intérieur des itérations mais le gain est faible bien que non négligeable lorsqu'on traite des données de grande taille. Grâce au travail sur le calcul de la distance, nous avons rendu possible la parcellisation des données d'IRMf sans réduction des données au préalable et en temps raisonnable comme le montre la table 10.1.

L'objectif était aussi de proposer une chaîne complète intégrant l'algorithme de parcellisation. C'est pourquoi, afin d'obtenir une parcellisation optimale, un soin particulier a été porté aux prétraitements appliqués sur les données. La normalisation des données a été perfectionnée en ajoutant une étape de recalage non linéaire (*warping*) individuel pour mieux corriger les distorsions géométriques. En moyennant les images fonctionnelles sur le groupe, notre méthode rend les contours entre tissus plus nets et recouvre plus de signal pour les coupes inférieures et

	Taille du problème	64 threads	Temps séquentiel
Parcellisation individuelle (substance grise)	932[1] × 233 539	38 min	5h54 min
Parcellisation individuelle (cerveau entier)	988×564 471	3h52 min	>48 h
Parcellisation de groupe (substance grise)	3 703[2]×155 697	26 min	8h21 min

TABLE 10.1 – Temps de calcul pour la parcellisation sur COMP2.

supérieures. En constatant que les mouvements du crâne sont faiblement corrélés aux perturbations du signal image, nous avons également développé une méthode de censure des acquisitions contaminées par le mouvement en se fondant uniquement sur les variations de signal dans la substance grise et dans la substance blanche. Pour cela, chaque acquisition a été synthétisée sous forme d'un nuage de points en mettant en ordonnée la variation de signal dans la substance grise et en abscisse celle dans la substance blanche. La méthode consiste ensuite à tracer une ellipse de confiance et à considérer comme aberrantes toutes les acquisitions en dehors de cette ellipse. Nous avons également montré qu'il faut s'affranchir de la ligne de base avant de parcelliser sous peine d'être dominé par des facteurs instrumentaux, plutôt que par l'information fonctionnelle. Nous avons proposé de faire une parcellisation de groupe en concaténant les signaux de chaque sujet, ce que permet l'algorithme dans la limite des moyens informatiques disponibles. Une fois la parcellisation obtenue, plusieurs indices ont été étudiés pour mesurer la qualité de la parcellisation. Ces indices suggèrent, comme attendu, que la parcellisation de groupe est plus cohérente. À l'aide d'un paradigme spécialement mis au point pour cette étude et mis en œuvre sur quatre sujets, nous avons également identifié des régions cérébrales activées à l'aide du coefficient de corrélation de Pearson et situées dans une partie du cortex somato-sensoriel et du cortex gustatif. Dans ces régions, des signaux fonctionnels de différentes formes ont été révélés, ce qui démontre la puissance de cette approche sans modèle.

10.2 Valorisations

Sont données ci-dessous les publications soumises en rapport avec la thèse.

[1] Mélodie Angeletti, Jean-Marie Bonny, Franck Durif, and Jonas Koko. Parallel Hierarchical Agglomerative Clustering for fMRI Data. In Roman Wyrzykowski, Jack Dongarra, Ewa Deelman, and Konrad Karczewski, editors, *Parallel Processing and Applied Mathematics*, pages 265–275, Cham, 2018. Springer International Publishing.

[2] Mélodie Angeletti, Jean-Marie Bonny, Franck Durif, and Jonas Koko. Parallelization of Ward's algorithm for fast clustering of fMRI datasets. Conférence Organisation of Human Brain mapping (OHBM 2018), Singapour, June 2018.

[3] Mélodie Angeletti, Jean-Marie Bonny, and Jonas Koko. Parallel Euclidean distance matrix computation on big datasets. en cours de soumission.

[4] Mélodie Angeletti, Jonas Koko, Franck Durif, and Jean-Marie Bonny. Model-free analysis of fMRI brain responses to food tastes using fast clustering. Conférence Organisation of Human Brain mapping (OHBM 2018), Singapour, June 2018.

[5] Mélodie Angeletti, Sylvie Clerjon, Jonas Koko, and Jean-Marie Bonny. Detection of robust group-level fMRI brain responses to food tastes. Présentation orale-14th International Conference on the Applications of Magnetic Resonance in Food Science 2018, sept 2018.

1. après censure avec une ellipse à 95%
2. concaténation du signal censuré avec une ellipse à 95% des quatre sujets

10.3 Perspectives concernant le calcul de la distance et l'algorithme de Ward

L'algorithme de Ward peut être appliqué à d'autres types de données en considérant que n_{T} est le nombre de caractéristiques relevées par observation et n_{V} le nombre d'observations total. Le calcul de la distance euclidienne représente une contribution majeure de la thèse. Toutefois, plusieurs points pourraient être étudiés voire améliorés.

10.3.1 Stockage de la matrice de distance

Tout d'abord, la section 7.2 propose deux façons de stocker la matrice de distance. Il est légitime de s'interroger sur l'influence du stockage sur les performances du calcul de la distance. En effet, le stockage de la distance détermine ce que le processeur charge en mémoire cache lorsqu'il range une distance et donc il doit être pris en compte pour exploiter au mieux la localité spatiale de la distance. La localité temporelle du vecteur de distance n'intervient pas car même lorsque la matrice $\boldsymbol{X}$ est découpée en lignes, la distance est calculée sur toutes les lignes avant d'être rangée en mémoire donc chaque case du vecteur distance n'est utilisé qu'une seule fois. Cependant, s'il faut tenir compte de la localité spatiale du vecteur de distance, il faut également exploiter au mieux la localité spatiale et temporelle de la matrice $\boldsymbol{X}$ utilisée pour calculer la distance. La manière de ranger la distance en mémoire devrait avoir un impact sur les performances en séquentiel et sur des architectures à mémoire partagée voire à mémoire distribuée. Cependant, pour le calcul sur GPU, l'influence devrait être moindre puisque le GPU dispose uniquement de deux blocs rectangulaires de la matrice $\boldsymbol{X}$ à la fois et que le rangement dans le vecteur distance est assuré par le CPU. De plus, le gain observé avec le GPU est essentiellement dû à la puissance de calcul du GPU.

10.3.2 Parallélisation hybride

10.3.2.1 Paralélisation MPI+OpenMP ou MPI+GPU

Puisque nous disposons d'un algorithme adapté aux architectures à mémoire distribuée, nous pouvons envisager de développer un algorithme pour des architectures hybrides. Par exemple, si l'un des nœuds de calcul possède plusieurs cœurs, la distance entre de deux tuiles peut être parallélisée avec OpenMP. De même si un des nœuds possède un GPU, il est envisageable de déléguer le calcul de la distance au GPU.

10.3.2.2 Calcul de la distance par découpage en blocs de lignes et de colonnes ($3^{ème}$ version) avec CUDA+OpenMP

Par manque de temps, nous n'avons pas développé le calcul de la distance avec un découpage des lignes et des colonnes en combinant CUDA+OpenMP. Une possibilité serait de combiner le déroulement de la boucle introduit en section 7.4.2 et le découpage des lignes. L'idée reviendrait donc à calculer la distance entre plusieurs paires de tuiles ($\boldsymbol{X}_{.i}$,$\boldsymbol{X}_{.j}$) à l'aide des *streams* en calculant la distance par bloc de lignes. La section 7.5.2.2 au lieu de dérouler la boucle délègue à deux *streams* distincts le calcul sur les blocs de lignes d'indice pair et le calcul sur les blocs de lignes d'indice impair. On peut combiner ce dédoublement au déroulement de la boucle avec pour seule limite la taille de la mémoire GPU. En effet, lorsqu'on déroule la boucle à une profondeur p, il faut allouer p fois plus de mémoires sur le GPU. Il faudrait ensuite évaluer si combiner le déroulement de la boucle et le dédoublement sur les blocs de lignes donne de meilleures performances que le déroulement de la boucle. On pourrait ensuite paralléliser le déroulement de la boucle à l'aide de OpenMP. Une autre solution serait d'utiliser OpenMP pour calculer la distance sur un bloc de la matrice de distance pendant que le GPU calcule. Cependant,il est

possible que la taille des blocs et des tuiles donnant les meilleures performances sur GPU et avec OpenMP ne soient pas la même. Il faudrait alors gérer des tuiles de taille différentes.

10.3.3 Précision du calcul de la distance

Lors du développement des différentes parallélisations, nous nous sommes assurés que les implémentations développées donnent des résultats égaux. Nous avons pour cela fixé le seuil d'erreur à 10^{-4} en double et simple précision. Il serait intéressant d'étudier les différences numériques des différentes versions plus en profondeur.

De plus, en simple précision, on s'est aperçu que suivant la taille des blocs choisie, les résultats ne sont pas corrects numériquement. Par exemple, en prenant chaque colonne de la matrice égale à l'indice de la colonne de sorte que $\boldsymbol{D}_{ij} = (i-j)^2 n_{\mathrm{T}}$, le calcul de la distance sur GPU en prenant m_R=64 et m_C=256 et en choisissant m_R=256 et m_C=256 ne donne pas le même résultat pour $\boldsymbol{D}(257, 259)$: le premier calcul trouve 4 000 qui est le résultat correct et le deuxième donne 3 976. Il serait intéressant d'identifier précisément d'où vient cette erreur numérique et voir s'il est possible de la corriger.

10.3.4 Extension aux distances L^p

Le chapitre 7 est consacré au calcul de la distance euclidienne. Cependant, il serait possible d'étendre le travail effectué au calcul de la distance L^p. Soit $\boldsymbol{x}$ un vecteur de $\mathbb{R}^n$, la norme associée à l'espace L^p,$p \in \mathbb{R}$, $p \geq 1$, notée $\|.\|_p$ est définie comme suit :

$$\|x\|_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{\frac{1}{p}} .$$

10.3.4.1 Calcul de la distance L^p sur des architectures à mémoire partagée ou à mémoire distribuée

Pour les architectures à mémoire partagée, la répartition des sous-matrices $\boldsymbol{D}_{ij}$ entre les différents processeurs reste inchangée. Cependant, plutôt que d'adapter l'algorithme 13 présenté en section 7.5, on propose ici d'adapter l'algorithme 12. En effet, dans l'algorithme 13, le calcul de la distance euclidienne s'effectue en deux étapes : le calcul de la norme puis le calcul du double-produit, ce qui est optimisé dans le cas de la distance euclidienne. Cependant, on veut traiter ici une distance L^p avec un $p \geq 1$ quelconque. Or, l'algorithme 12 calcule la distance en déroulant la boucle sur les lignes (**cf. les boucles en lignes 10, 16, 27 et 33**). Ces boucles sont adaptables à la norme $\|.\|_p$ en prenant la valeur absolue et en élevant à la puissance p au lieu de deux. L'algorithme 15 montre l'adaptation de l'algorithme 10 pour la distance L^p. On a changé l'expression du calcul de la distance dans les boucles en lignes 12, 19, 27 et 33. Avant la boucle finale commençant en ligne 35, on a calculé la norme $\|.\|_p$ à la puissance p : la boucle en ligne 35 permet alors de calculer la norme $\|.\|_p$.

Sur des architectures à mémoire distribuée, on a considéré en section 7.5.2.3 qu'on pouvait calculer la distance euclidienne de manière efficace et ensuite on a défini le schéma de communication entre les processeurs. Dans le cas de la distance L^p, le schéma de communication reste identique, ce qui change est le calcul de la distance. Sur chaque processeur, on peut appliquer l'algorithme 15.

10.3.4.2 Calcul de la distance L^p sur GPU

L'implémentation de l'algorithme 13 sur GPU est fondée sur l'utilisation des routines BLAS sur GPU et donc sur du calcul matriciel (**cf. Section 7.5.2.2**). Par conséquent, nous ne pouvons pas adapter l'algorithme CUDA de la même manière que nous avons modifié les algorithmes en OpenMP et MPI en section 10.3.4.1.

Algorithme 15 : Calcul de la distance L^p

Entrée : $\boldsymbol{X}$ matrice de taille $n_{\mathrm{T}} \times n_{\mathrm{V}}$; m_R nombre de lignes par bloc ; m_C nombre de colonnes par tuile

Sortie : $\boldsymbol{D}$ vecteur de taille $\frac{n_{\mathrm{V}}(n_{\mathrm{V}}-1)}{2}$

$n_{CB} = \left\lceil \frac{n_{\mathrm{V}}}{m_C} \right\rceil$; $n_{RB} = \left\lceil \frac{n_{\mathrm{T}}}{m_R} \right\rceil$; $n_B = \frac{n_{CB}[n_{CB}+1)}{2}$;

pour *κ de 1 à n_B* **faire**

  $K = \kappa + \frac{n_{CB}(1-n_{CB})}{2} + \frac{1}{2} \left\lfloor \frac{\sqrt{8\left(\frac{n_{CB}(n_{CB}+1)}{2} - \kappa\right)+1}-1}{2} \right\rfloor \left(\left\lfloor \frac{\sqrt{8\left(\frac{n_{CB}(n_{CB}+1)}{2} - \kappa\right)+1}-1}{2} \right\rfloor + 1 \right)$;

  $L = n_{CB} - \left\lfloor \frac{\sqrt{8\left(\frac{n_{CB}(n_{CB}+1)}{2} - \kappa\right)+1}-1}{2} \right\rfloor$;

  i_1=1+(K-1)m_C ; i_2=Km_C ; j_1=1+(L-1)m_C ; j_2=Lm_C;

  iter=1 ; m_1=1+(iter-1)m_R ; m_2=iter $\times$ m_R;

  si $K > L$ **alors**

    pour *i de i_1 à i_2* **faire**

      pour *j de j_1 à j_2* **faire**

        $k = j + \frac{(i-1)(i-2)}{2}$;

        d(k)=$(x_{m_1 i} - x_{m_2 j})^2$;

        pour *m de $m_1 + 1$ à m_2* **faire**

          d(k)=d(k)+$|x_{mi} - x_{mj}|^p$;

  sinon

    pour *j de j_1 à j_2* **faire**

      pour *i de i_1 à $j - 1$* **faire**

        $k = i + \frac{(j-1)(j-2)}{2}$;

        d(k)=$(x_{m_1 i} - x_{m_2 j})^2$;

        pour *m de $m_1 + 1$ à m_2* **faire**

          d(k)=d(k)+$|x_{mi} - x_{mj}|^p$;

  pour *iter de 2 à n_{RB}* **faire**

    m_1=1+(iter-1)m_R ; m_2=iter $\times$ m_R;

    si $K > L$ **alors**

      pour *i de i_1 à i_2* **faire**

        pour *j de j_1 à j_2* **faire**

          $k = j + \frac{(i-1)(i-2)}{2}$;

          pour *m de m_1 à m_2* **faire**

            d(k)=d(k)+$|x_{mi} - x_{mj}|^p$;

      sinon

        pour *j de j_1 à j_2* **faire**

          pour *i de i_1 à $j - 1$* **faire**

            $k = i + \frac{(j-1)(j-2)}{2}$;

            pour *m de m_1 à m_2* **faire**

              d(k)=d(k)+$|x_{mi} - x_{mj}|^p$;

pour *k de 1 à $\frac{n_{\mathrm{V}}(n_{\mathrm{V}}-1)}{2}$* **faire**

  d(k)=$\mathrm{d(k)}^{\frac{1}{p}}$;

Dans le cas où on considère $p \in \mathbb{N}*$ pair, on peut étendre le calcul de la distance euclidienne à la distance L^p en utilisant CUBLAS. Considérons deux tuiles rectangulaires de la matrice $\boldsymbol{X}$: la tuile $\boldsymbol{X}_{.i}$ contenant les colonnes d'indice i_1 à i_m et la tuile $\boldsymbol{X}_{.j}$ contenant les colonnes d'indice j_1 à j_n, l'expression de la distance (élevée à la puissance p) entre chaque paire de colonnes (i, j) avec $i_1 \leq i \leq i_m$, $j_1 \leq j \leq j_n$ est donné par :

$$\boldsymbol{D}(\boldsymbol{X}_i, \boldsymbol{X}_j) = \begin{pmatrix} \sum_{l=1}^{n_\mathrm{T}} |x_{li_1} - x_{lj_1}|^p & \dots & \sum_{l=1}^{n_\mathrm{T}} |x_{li_1} - x_{lj_n}|^p \\ \vdots & \ddots & \vdots \\ \sum_{l=1}^{n_\mathrm{T}} |x_{li_m} - x_{lj_1}|^p & \dots & \sum_{l=1}^{n_\mathrm{T}} |x_{li_m} - x_{lj_n}|^p \end{pmatrix} \tag{10.1}$$

Puisqu'on travaille dans le cas où p est pair, on peut s'affranchir des valeurs absolues et donc développer l'expression de la distance donnée en (10.1) en utilisant la formule du binôme de Newton. On obtient alors :

$$\begin{aligned} \boldsymbol{D}(\boldsymbol{X}_{.i}, \boldsymbol{X}_{.j}) &= \begin{pmatrix} \sum_{l=1}^{n_\mathrm{T}} \sum_{k=0}^{p} \binom{n}{k} (x_{li_1})^{p-k} (-x_{lj_1})^k & \dots & \sum_{l=1}^{n_\mathrm{T}} \sum_{k=0}^{p} \binom{n}{k} (x_{li_1})^{p-k} (-x_{lj_n})^k \\ \vdots & \ddots & \vdots \\ \sum_{l=1}^{n_\mathrm{T}} \sum_{k=0}^{p} \binom{n}{k} (x_{li_m})^{p-k} (-x_{lj_1})^k & \dots & \sum_{l=1}^{n_\mathrm{T}} \sum_{k=0}^{p} \binom{n}{k} (x_{li_m})^{p-k} (-\boldsymbol{x}_{lj_n})^k \end{pmatrix} \\ &= \begin{pmatrix} \sum_{k=0}^{p} \binom{n}{k} \sum_{l=1}^{n_\mathrm{T}} (x_{li_1})^{p-k} (-x_{lj_1})^k & \dots & \sum_{k=0}^{p} \binom{n}{k} \sum_{l=1}^{n_\mathrm{T}} (x_{li_1})^{p-k} (-x_{lj_n})^k \\ \vdots & \ddots & \vdots \\ \sum_{k=0}^{p} \binom{n}{k} \sum_{l=1}^{n_\mathrm{T}} (x_{li_m})^{p-k} (-xt_{lj_1})^k & \dots & \sum_{k=0}^{p} \binom{n}{k} \sum_{l=1}^{n_\mathrm{T}} (x_{li_m})^{p-k} (-x_{lj_n})^k \end{pmatrix} \\ &= \begin{pmatrix} \sum_{k=0}^{p} \binom{n}{k} (-1)^k \sum_{l=1}^{n_\mathrm{T}} (x_{li_1})^{p-k} (x_{lj_1})^k & \dots & \sum_{k=0}^{p} \binom{n}{k} (-1)^k \sum_{l=1}^{n_\mathrm{T}} (x_{li_1})^{p-k} (x_{lj_n})^k \\ \vdots & \ddots & \vdots \\ \sum_{k=0}^{p} \binom{n}{k} (-1)^k \sum_{l=1}^{n_\mathrm{T}} (x_{li_m})^{p-k} (x_{lj_1})^k & \dots & \sum_{k=0}^{p} \binom{n}{k} (-1)^k \sum_{l=1}^{n_\mathrm{T}} (x_{li_m})^{p-k} (x_{lj_n})^k \end{pmatrix} \end{aligned} \tag{10.2}$$

Supposons maintenant qu'on dispose également de l'opération $\hat{}^k$ telle que pour la matrice $\boldsymbol{A}$ et pour un k quelconque, on obtienne :

$$\boldsymbol{A}\hat{}^k = \begin{pmatrix} x_{1i_1}^k & \dots & x_{1i_m}^k \\ \vdots & \ddots & \vdots \\ x_{n_\mathrm{T}i_1}^k & \dots & x_{n_\mathrm{T}i_m}^k \end{pmatrix} \quad .$$

Soit maintenant α et β deux réels, calculons le produit matriciel $\boldsymbol{A}\hat{}^{\alpha\top}\boldsymbol{B}\hat{}^{\beta}$:

$$\begin{aligned} \boldsymbol{A}\hat{}^{\alpha\top}\boldsymbol{B}\hat{}^{\beta} &= \begin{pmatrix} x_{1i_1}^k & \dots & x_{n_\mathrm{T}i_1}^k \\ \vdots & \ddots & \vdots \\ x_{1i_m}^k & \dots & x_{n_\mathrm{T}i_m}^k \end{pmatrix} \times \begin{pmatrix} x_{1j_1}^k & \dots & x_{1j_m}^k \\ \vdots & \ddots & \vdots \\ xt_{n_\mathrm{T}j_1}^k & \dots & x_{n_\mathrm{T}j_m}^k \end{pmatrix} \\ &= \begin{pmatrix} \sum_{l=1}^{n_\mathrm{T}} x_{li_1}^\alpha x_{lj_1}^\beta & \dots & \sum_{l=1}^{n_\mathrm{T}} x_{li_1}^\alpha x_{lj_n}^\beta \\ \vdots & \ddots & \vdots \\ \sum_{l=1}^{n_\mathrm{T}} x_{li_m}^\alpha x_{lj_1}^\beta & \dots & \sum_{l=1}^{n_\mathrm{T}} x_{li_m}^\alpha x_{lj_n}^\beta \end{pmatrix} \end{aligned} \tag{10.3}$$

En prenant $\alpha = p - k$,$\beta = k$,$\boldsymbol{A} = \boldsymbol{X}_{.i}$ et $\boldsymbol{B} = \boldsymbol{X}_{.j}$, on retrouve la seconde somme de l'expression de $\boldsymbol{D}(\boldsymbol{X}_{.i},\boldsymbol{X}_{.j})$ donnée en équation (10.2). Donc, on peut écrire la distance comme une somme de produit matriciel :

$$\boldsymbol{D}(\boldsymbol{X}_{.i}, \boldsymbol{X}_{.j}) = \sum_{k=0}^{p} \binom{n}{k} (-1) k \boldsymbol{X}_{.i}\hat{}^{p-k\top} \boldsymbol{X}_{.j}\hat{}^{k} \quad .$$

Décrivons maintenant comment on peut l'implémenter sur GPU. On peut créer un noyau CUDA correspondant à l'opération $\hat{}^k$ en demandant à chaque thread d'élever à la puissance k un élément de la matrice. Le noyau aura sans doute besoin de charger la matrice dans sa mémoire partagée pour limiter les temps d'accès mémoire. Pour stocker $\boldsymbol{X}_{.i}\hat{}^{p-k}$ et $\boldsymbol{X}_{.j}\hat{}^{k}$, le GPU a besoin d'avoir deux tuiles supplémentaires en mémoire, car il faut garder $\boldsymbol{X}_{.i}$ et $\boldsymbol{X}_{.j}$ en mémoire pour ne pas avoir à les copier depuis le CPU à chaque fois. Il faut également disposer des coefficients binomiaux : on peut calculer les coefficients binomiaux sur CPU et les stocker dans un tableau qu'on transmet au GPU mais Altaf Wani *et al.* proposent une implémentation du calcul des coefficients binomiaux sur GPU [7]. Lorsqu'on prend p entier strictement positif pair ; on peut s'affranchir de la valeur absolue. En revanche, pour p entier strictement positif impair, la valeur absolue ne peut pas être supprimée et le calcul sous forme de produit matriciel n'est plus valable.

Si on veut étendre le calcul de la distance euclidienne à n'importe quel $p \in \mathbb{R}$, il faut alors écrire un noyau spécifique. Pour cela, on peut s'inspirer de ce qu'ont proposé Kim *et al.* dans [117]. Leur approche ne permet pas toutefois de traiter le cas de matrices de très grandes tailles. Cependant, en découpant la matrice $\boldsymbol{X}$ en tuiles rectangulaires et en appelant leur noyau adapté au cas des distances L^p, il doit être possible de calculer la distance pour de grandes matrices. L'algorithme 16 associe le découpage en tuiles de la matrice $\boldsymbol{X}$ avec l'appel à un noyau CUDA calculant la distance L^p (élevée à la puissance p) donné par le listing 37. L'algorithme commence par calculer le nombre de tuiles et le nombre de blocs (autrement dit le nombre de paires de tuiles) nécessaires en ligne 1 ; puis il fait les allocations sur GPU en ligne 2 pour stocker sur le GPU les copies des tuiles $\boldsymbol{X}_{.i}$ et $\boldsymbol{X}_{.j}$ et en ligne 3 pour stocker la sous-matrice de distance calculée sur GPU. L'algorithme itère ensuite sur les blocs (boucle commençant en ligne 4). Pour chaque bloc, on calcule les deux indices de tuile auquel il correspond en lignes 5 et 6. Une fois les tuiles identifiées, le CPU les copie dans des tuiles intermédiaires aux lignes 7 et 9, puis il copie ces matrices sur le GPU en lignes 8 et 10. Le CPU lance ensuite l'appel au noyau CUDA pour calculer la distance L^p (qui ne s'exécute que lorsque les deux tuiles aient été copiées entièrement sur le GPU) en ligne 11. Une fois que l'exécution du noyau est terminée, le GPU copie la matrice résultante $\boldsymbol{M}$ sur le CPU (ligne 12). Lorsque la copie est terminée, le CPU peut ranger la matrice résultante dans le vecteur de distance en ligne 13.

```
__global__ void distanceLp(int nb_rows,int nb_cols,double *A,double
    *B,int p,double *out){
__shared__ double Xs[BlockSize][BlockSize];
__shared__ double Ys[BlockSize][BlockSize];
int bx=blockIdx.x; int by=blockIdx.y;
int tx=threadIdx.x; int ty=threadIdx.y;
//indexes of the sub-matrices
int xBeg=bx*BlockSize*nb_rows;
int yBeg=by*BlockSize*nb_rows;
int xEnd=xBeg+nb_rows;
//loop indexes
int x,y,k;
//out index
int outInd;
//temporary variable
double temp;
//euclidian distance
double dist=0.0;

//loop on the tile to have all the rows
 for (x=xBeg,y=yBeg;x<xEnd;x+=BlockSize,y+=BlockSize){
  //each thread fills one element of sub-matrix
```

```
    Ys[tx][ty]=B[y+ty*nb_rows+tx];
    Xs[tx][ty]=A[x+ty*nb_rows+tx];
    //syncrhonize all threads to have complete sub_matrices
    __syncthreads();
    //loop on the rows of submatrix
    for (k=0;k<BlockSize;k++){
     temp=fabs(Xs[k][tx]-Ys[k][ty]);
     dist=dist+pow(temp,p);
    }
    __syncthreads();
  }
 //each thread computes one case of distance
 outInd=ty+nb_cols*tx+bx*BlockSize*nb_cols+by*BlockSize;
 out[outInd]=dist;
}
```

Listing 10.1 – Noyau CUDA pour le calcul de la distance L^p.

Algorithme 16 : Calcul de la distance L^p sur GPU

Entrée : $\boldsymbol{X}$ matrice de taille $n_\mathrm{T} \times n_\mathrm{V}$;
m_C le nombre de colonnes par tuile
Sortie : $\boldsymbol{D}$ vecteur de taille $\frac{n_\mathrm{V}(n_\mathrm{V}-1)}{2}$

1 $n_{CB} = \left\lceil \frac{n_\mathrm{V}}{m_C} \right\rceil$; $n_B = \frac{n_{CB}[n_{CB}+1)}{2}$;
2 Allouer sur le GPU $\boldsymbol{X}_{.i}$ et $\boldsymbol{X}_{.j}$ de taille $n_\mathrm{T} \times m_C$;
3 Allouer sur le GPU $\boldsymbol{M}$ de taille $m_C \times m_C$;
4 **pour** κ *de 1 à* n_B **faire**
5 $K = \kappa + \frac{n_{CB}(1-n_{CB})}{2} + \frac{1}{2}\left\lfloor \frac{\sqrt{8\left(\frac{n_{CB}(n_{CB}+1)}{2}-\kappa\right)+1}-1}{2} \right\rfloor \left(\left\lfloor \frac{\sqrt{8\left(\frac{n_{CB}(n_{CB}+1)}{2}-\kappa\right)+1}-1}{2} \right\rfloor + 1 \right)$;
6 $L = n_{CB} - \left\lfloor \frac{\sqrt{8\left(\frac{n_{CB}(n_{CB}+1)}{2}-\kappa\right)+1}-1}{2} \right\rfloor$;
7 Calculer $\boldsymbol{X}_{.K}$ sur CPU;
8 Copier $\boldsymbol{X}_{.K}$ dans $\boldsymbol{X}_{.i}$ sur le GPU;
9 Calculer $\boldsymbol{X}_{.L}$ sur CPU;
10 Copier $\boldsymbol{X}_{.L}$ dans $\boldsymbol{X}_{.j}$ sur le GPU;
11 Lancer le noyau distanceLp(n_T,m_C,$\boldsymbol{X}_{.i}$,$\boldsymbol{X}_{.j}$,p,$\boldsymbol{M}$);
12 Copier $\boldsymbol{M}$ sur le CPU;
13 $\boldsymbol{D}_{tiM}$=$\boldsymbol{M}$;

10.3.5 Parallélisation des itérations

Au sein des itérations, nous avons parallélisé uniquement la mise à jour de la distance et des vecteurs ***minDist*** et ***minLoc***. Cependant, le gain ainsi obtenu est faible. Nous pourrions paralléliser également la recherche de la distance minimale, qui peut se faire à l'aide d'une opération de réduction. Pour cela, plusieurs auteurs recherchent les plus proches voisins ou trient la distance avant d'exécuter les itérations [175, 159, 125]. Zhang *et al.* ont réussi à implémenter les itérations sur GPU mais ils n'ont testé leur implémentation que sur des petites données[233]. Ce sont des pistes à explorer plus en détail pour améliorer notre implémentation des itérations de l'algorithme de Ward.

10.4 Perspectives pour l'analyse des données d'IRMf

10.4.1 Accessibilité du code pour la communauté d'IRMf

Les codes permettant d'obtenir les résultats présentés dans ce manuscrit ont été codé soit en MATLAB soit en Fortran : les prétraitements des données sont faits notamment à l'aide de SPM12 donc sous MATLAB, l'algorithme de Ward est quant à lui codé en Fortran (voire en langage C pour la partie GPU), et les post-traitements pour obtenir, par exemple la parcellisation sous forme d'images 3D, sont en MATLAB. Actuellement, le passage de MATLAB vers Fortran s'effectue en convertissant les données en une matrice qui est stockée dans un fichier texte. Les résultats de la parcellisation sont des vecteurs également stockés dans des fichiers textes pouvant être lus par MATLAB.

L'idéal pour l'utilisateur serait de limiter l'usage d'un terminal et de passer essentiellement par MATLAB. En effet, MATLAB reste très usité par la communauté des neuro-scientifiques bien qu'une partie de la communauté tente de migrer vers des logiciels OpenSource, contrairement à Fortran. Or, il est possible d'appeler un code Fortran dans MATLAB : soit en demandant à MATLAB d'exécuter le code Fortran dans un terminal, soit de créer un MEX-File qui permet d'appeler un programme Fortran dans MATLAB.

10.4.2 Évaluation de la méthode de censure

La spécificité des données obtenues avec ce type de stimulation « naturelle » est de générer d'importants artéfacts de mouvement, dont l'amplitude et la fréquence dépend beaucoup du sujet. C'est pourquoi, la « correction » de mouvement (terme qui s'avère impropre dans le cas d'une censure qui enlève les données impropres) est une étape essentielle. Nous avons développé une méthode de censure originale fondée sur la détection des variations aberrantes du signal à l'aide d'une ellipse de confiance. L'intervalle de confiance a été déterminé à partir des diagrammes de Power à l'aide d'une inspection visuelle. Cependant, l'influence de cet intervalle de confiance sur la parcellisation mériterait d'être étudié. On peut néanmoins anticiper certains aspects. Si on réduit l'intervalle de confiance (passant de 95% à 80%), le nombre d'acquisitions censurées augmente et la puissance statistique de l'expérience diminue. La censure réduit donc les faux positifs liés au mouvement, tout en réduisant la capacité de détection de l'information fonctionnelle. Cette analyse en fonction du niveau de confiance pourrait être effectuée en comparant les résultats de parcellisation. Pour autant, compte tenu des difficultés d'interprétation de la parcellisation, il serait sans doute plus simple de comparer les résultats obtenus par GLM classique ou MVPA. De plus, il serait intéressant de comparer notre méthode de censure aux méthodes classiques s'appuyant sur les paramètres de mouvement ou sur une analyse en composantes indépendantes. Comparer deux modes de censure permettrait d'évaluer quelles acquisitions sont censurées par notre méthode sans être censurées l'autre et inversement. In fine la principale difficulté à comparer des inférences fonctionnelles réside dans l'absence de *gold standard*. Il faut s'appuyer sur des critères statistiques variés pour valider la pertinence d'une méthode par rapport à une autre (cf. [186] par exemple).

10.4.3 Parcellisation : perspectives

L'objectif de la thèse était essentiellement de se donner les moyens de mener une analyse guidée par les données d'IRMf de stimuli alimentaires. La contrainte était aussi d'introduire le minimum d'*a priori*. Une solution intégrée est proposée et il est possible maintenant de l'appliquer à un large spectre de jeux de données.

Cela étant, la parcellisation est une approche encore peu populaire en IRMf. Elle n'émerge que maintenant, comme en témoigne le numéro spécial de Neuroimage qui lui a été consacré cette année. À l'inverse de l'analyse GLM classique fondée sur un modèle qui ne fait apparaître que les territoires dont les réponses correspondent aux modèles, la parcellisation produit énormément

de données puisque tous les voxels appartiennent à une parcelle. La principale difficulté réside maintenant dans l'exploitation de cette information multi-échelle.

Des tentatives ont été menées dans ce sens en identifiant les parcelles d'intérêt à l'aide de la corrélation avec un modèle. Cette approche, bien que fructueuse, présente plusieurs inconvénients dont le principal est de nécessiter un modèle. C'est pourquoi l'approche la plus naturelle est d'établir des corrélations entre parcelles, comme montré dans le chapitre 9. Même si les corrélations trouvées avec notre jeu de données sont particulièrement fortes, ajouter un test de significativité serait nécessaire à l'aide de tests non paramétriques fondés sur la permutation. L'avantage de cette approche serait de ne garder qu'un nombre réduit de parcelles. Si le nombre de parcelles significatives reste élevé, une solution reste de sélectionner les parcelles à partir d'un critère neuro-anatomique. Une dernière option serait de mesurer l'amplitude de la réponse dans les parcelles. Ceci permettrait de trier les parcelles en fonction d'un critère simple à appréhender, ce qui permettrait en outre de sélectionner l'échelle pertinente.

Annexes

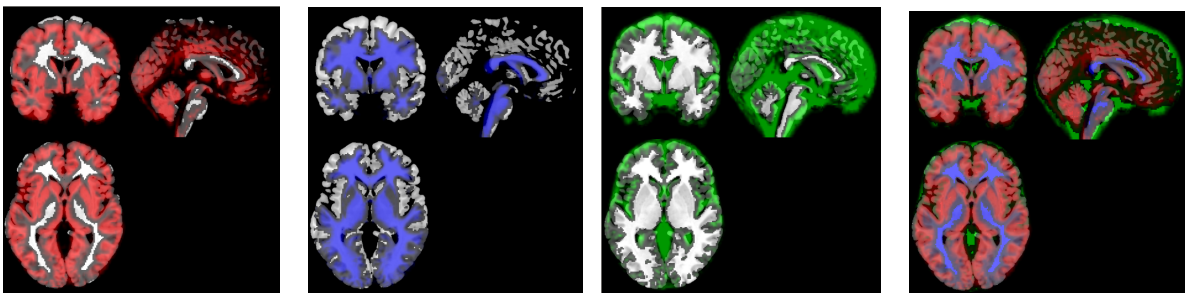
Annexe A

Anatomie du cerveau

Cette annexe permet d'identifier les différents tissus cérébraux : substance grise, substance blanche et liquide céphalo-rachidien ainsi que les régions d'intérêt issues de l'atlas AAL2.

A.1 Segmentation des tissus cérébraux

Lorsqu'on segmente l'image anatomique pondérée en T1 avec la fonction SEGMENT de SPM12, on obtient des cartes de probabilités d'appartenance. Pour chaque voxel, on dispose donc de sa probabilité d'être dans chacun des tissus considérés (substance grise, substance blanche, liquide céphalo-rachidien, os, tissu souple, air/fond). Les trois tissus qui nous intéressent sont la substance grise, la substance blanche et le liquide céphalo-rachidien, montrés en figure A.1.

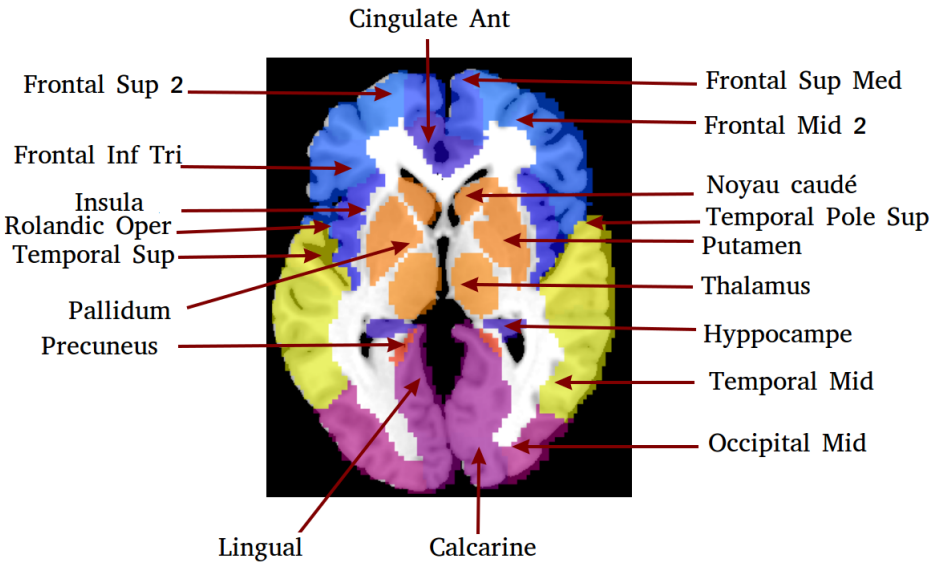


(a) Substance grise (b) Substance blanche (c) Liquide céphalo-rachidien (d) Les trois tissus

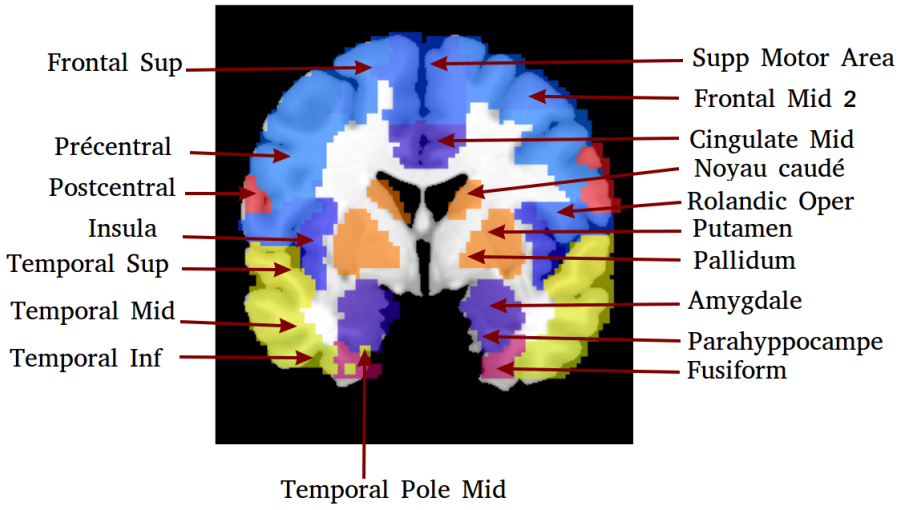
FIGURE A.1 – Segmentation.

A.2 Atlas AAL2

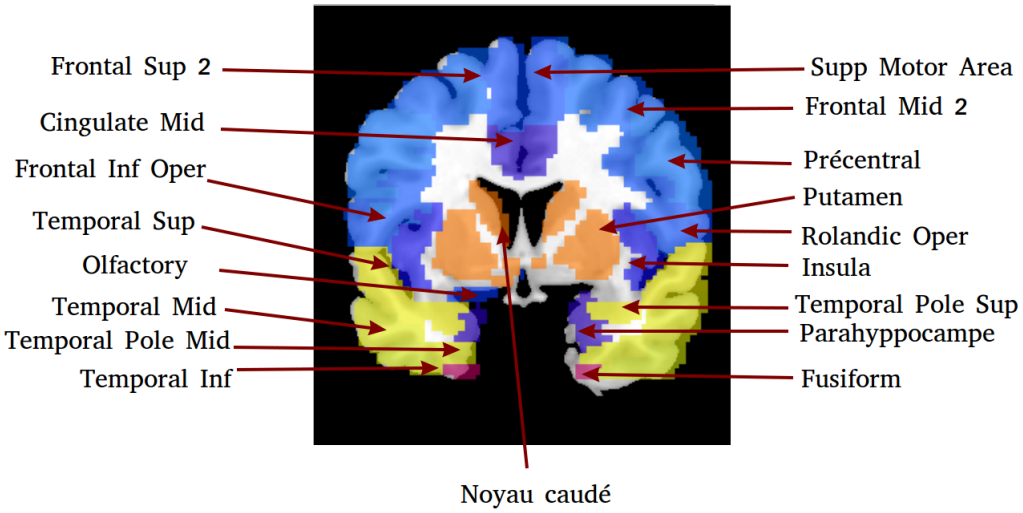
L'identification des parcelles se fait à l'aide de l'atlas AAL2. Les figures A.2 et A.3 montrent différentes coupes de l'atlas AAL2 superposé au template ch2better. Pour les coupes axiales et coronales, la symétrie par rapport aux hémisphères n'est pas montrée. Seule une partie des 120 régions d'intérêt est montrée.



(a) Coupe axiale z=0

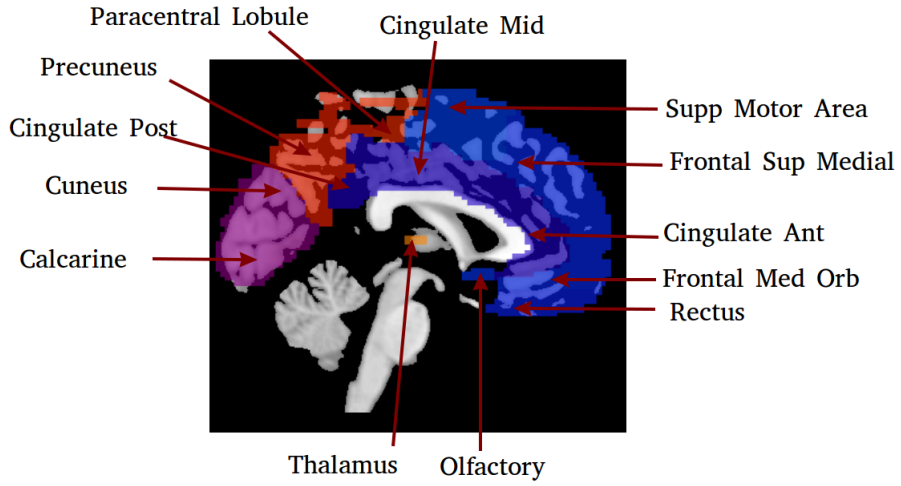


(b) Coupe coronale y=0

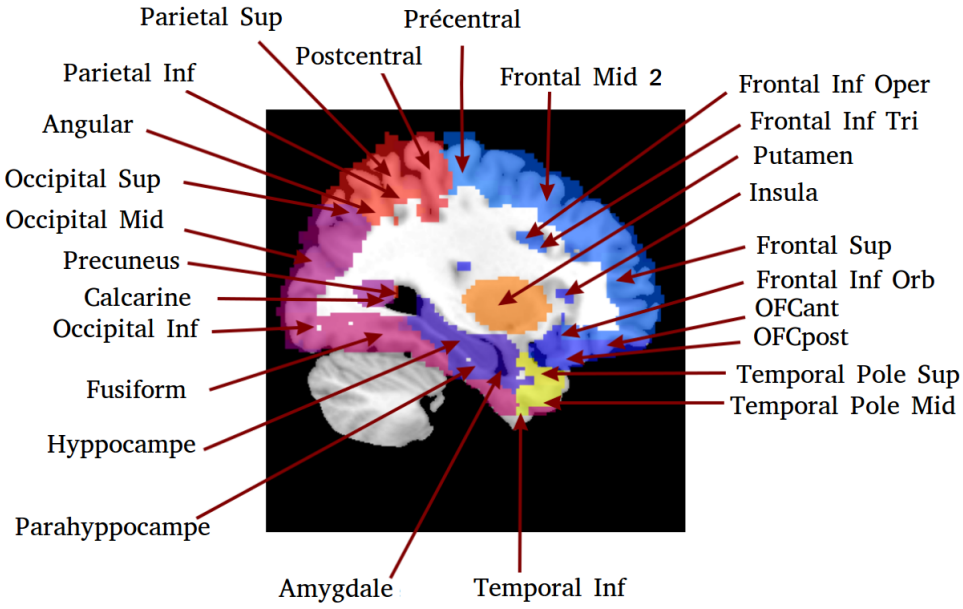


(c) Coupe coronale y=7

FIGURE A.2 – Atlas AAL : identification de quelques régions d'intérêt.



(d) Coupe sagittale x=0



(e) Coupe sagittale x=33

FIGURE A.3 – Atlas AAL : identification de quelques régions d'intérêt.

Annexe B

Démonstration de la formule de mise à jour de Lance-Williams pour l'algorithme de Ward

On rappelle que la distance de Ward est donnée par l'équation B.1 où C_i^k représente le cluster i à l'itération k, $\overline{C_i^k}$ son centroïde qui est un vecteur de dimension n_{T} et $\left|C_i^k\right|$ le nombre d'éléments dans le cluster . La formule de Lance-Williams permet de mettre à jour les distances entre le cluster $C_{i\cup j}^{k+1} = C_i^k \cup C_j^k$ issu de la fusion des clusters C_i^k et C_j^k avec un cluster C_l^k . Elle est donnée à l'équation B.2.

$$d\left(C_i^k, C_j^k\right) = \frac{\left|C_i^k\right| \left|C_j^k\right|}{\left|C_i^k\right| + \left|C_j^k\right|} \left\|\overline{C_i^k} - \overline{C_j^k}\right\|_2^2 \tag{B.1}$$

$$\begin{aligned} d\left(C_{i\cup j}^{k+1}, C_l^{k+1}\right) = & \frac{\left|C_i^k\right| + \left|C_l^k\right|}{\left|C_i^k\right| + \left|C_j^k\right| + \left|C_l^k\right|} d\left(C_i^k, C_l^k\right) + \frac{\left|C_j^k\right| + \left|C_l^k\right|}{\left|C_i^k\right| + \left|C_j^k\right| + \left|C_l^k\right|} d\left(C_j^k, C_l^k\right) \\ & - \frac{\left|C_l^k\right|}{\left|C_i^k\right| + \left|C_j^k\right| + \left|C_l^k\right|} d\left(C_i^k, C_j^k\right) \end{aligned} \tag{B.2}$$

Développons l'équation B.2 :

$$\begin{aligned} d\left(C_{i\cup j}^{k+1}, C_l^{k+1}\right) = & \frac{\left|C_i^k\right| + \left|C_l^k\right|}{\left|C_i^k\right| + \left|C_j^k\right| + \left|C_l^k\right|} \frac{\left|C_i^k\right| \left|C_l^k\right|}{\left|C_i^k\right| + \left|C_l^k\right|} \left\|\overline{C_i^k} - \overline{C_l^k}\right\|_2^2 \\ & + \frac{\left|C_j^k\right| + \left|C_l^k\right|}{\left|C_i^k\right| + \left|C_j^k\right| + \left|C_l^k\right|} \frac{\left|C_j^k\right| \left|C_l^k\right|}{\left|C_j^k\right| + \left|C_l^k\right|} \left\|\overline{C_j^k} - \overline{C_l^k}\right\|_2^2 \\ & - \frac{\left|C_l^k\right|}{\left|C_i^k\right| + \left|C_j^k\right| + \left|C_l^k\right|} \frac{\left|C_i^k\right| \left|C_j^k\right|}{\left|C_j^k\right| + \left|C_i^k\right|} \left\|\overline{C_i^k} - \overline{C_j^k}\right\|_2^2 \end{aligned}$$

$$
\begin{aligned}
d\left(C_{i\cup j}^{k+1}, C_l^{k+1}\right) = & \frac{\left|C_i^k\right|\left|C_l^k\right|}{\left|C_i^k\right| + \left|C_j^k\right| + \left|C_l^k\right|} \left\|\overline{C_i^k} - \overline{C_l^k}\right\|_2^2 \\
& + \frac{\left|C_j^k\right|\left|C_l^k\right|}{\left|C_i^k\right| + \left|C_j^k\right| + \left|C_l^k\right|} \left\|\overline{C_i^k} - \overline{C_l^k}\right\|_2^2 \\
& - \frac{\left|C_l^k\right|\left|C_i^k\right|\left|C_j^k\right|}{\left(\left|C_i^k\right| + \left|C_j^k\right| + \left|C_l^k\right|\right)\left(\left|C_j^k\right| + \left|C_i^k\right|\right)} \left\|\overline{C_i^k} - \overline{C_j^k}\right\|_2^2
\end{aligned} \tag{B.3}
$$

Comme $\forall (i\,j), \left\|\overline{C_i^k} - \overline{C_j^k}\right\|_2^2 = \sum_{t=1}^{n_\mathrm{T}} \left(\overline{C_i^k}(t) - \overline{C_j^k}(t)\right)^2$, on peut développer la formule B.3 en fonction de t.

$$
\begin{aligned}
d\left(C_{i\cup j}^{k+1}, C_l^{k+1}\right) = & \frac{\left|C_i^k\right|\left|C_l^k\right|}{\left|C_i^k\right| + \left|C_j^k\right| + \left|C_l^k\right|} \sum_{t=1}^{n_\mathrm{T}} \left(\overline{C_i^k}(t) - \overline{C_l^k}(t)\right)^2 + \frac{\left|C_j^k\right|\left|C_l^k\right|}{\left|C_i^k\right| + \left|C_j^k\right| + \left|C_l^k\right|} \sum_{t=1}^{n_\mathrm{T}} \left(\overline{C_j^k}(t) - \overline{C_l^k}(t)\right)^2 \\
& - \frac{\left|C_l^k\right|\left|C_i^k\right|\left|C_j^k\right|}{\left(\left|C_i^k\right| + \left|C_j^k\right| + \left|C_l^k\right|\right)\left(\left|C_j^k\right| + \left|C_i^k\right|\right)} \sum_{t=1}^{n_\mathrm{T}} \left(\overline{C_i^k}(t) - \overline{C_j^k}(t)\right)^2 \\
d\left(C_{i\cup j}^{k+1}, C_l^{k+1}\right) = & \left(\frac{\left|C_i^k\right|\left|C_l^k\right|}{\left|C_i^k\right| + \left|C_j^k\right| + \left|C_l^k\right|} - \frac{\left|C_l^k\right|\left|C_i^k\right|\left|C_j^k\right|}{\left(\left|C_i^k\right| + \left|C_j^k\right| + \left|C_l^k\right|\right)\left(\left|C_j^k\right| + \left|C_i^k\right|\right)}\right) \sum_{t=1}^{n_\mathrm{T}} \overline{C_i^k}(t)^2 \\
& + \left(\frac{\left|C_j^k\right|\left|C_l^k\right|}{\left|C_i^k\right| + \left|C_j^k\right| + \left|C_l^k\right|} - \frac{\left|C_l^k\right|\left|C_i^k\right|\left|C_j^k\right|}{\left(\left|C_i^k\right| + \left|C_j^k\right| + \left|C_l^k\right|\right)\left(\left|C_j^k\right| + \left|C_i^k\right|\right)}\right) \sum_{t=1}^{n_\mathrm{T}} \overline{C_j^k}(t)^2 \\
& + \frac{\left(\left|C_i^k\right| + \left|C_j^k\right|\right)\left|C_l^k\right|}{\left|C_i^k\right| + \left|C_j^k\right| + \left|C_l^k\right|} \sum_{t=1}^{n_\mathrm{T}} \overline{C_l^k}(t)^2 - 2\frac{\left|C_i^k\right|\left|C_l^k\right|}{\left|C_i^k\right| + \left|C_j^k\right| + \left|C_l^k\right|} \sum_{t=1}^{n_\mathrm{T}} \overline{C_i^k}(t)\overline{C_l^k}(t) \\
& - 2\frac{\left|C_j^k\right|\left|C_l^k\right|}{\left|C_i^k\right| + \left|C_j^k\right| + \left|C_l^k\right|} \sum_{t=1}^{n_\mathrm{T}} \overline{C_j^k}(t)\overline{C_l^k}(t) \\
& + 2\frac{\left|C_l^k\right|\left|C_i^k\right|\left|C_j^k\right|}{\left(\left|C_i^k\right| + \left|C_j^k\right| + \left|C_l^k\right|\right)\left(\left|C_j^k\right| + \left|C_i^k\right|\right)} \sum_{t=1}^{n_\mathrm{T}} \overline{C_i^k}(t)\overline{C_j^k}(t)
\end{aligned}
$$

$$
\begin{aligned}
d\left(C_{i\cup j}^{k+1}, C_l^{k+1}\right) = & \left(\frac{\left|C_i^k\right|^2 \left|C_l^k\right|}{\left(\left|C_i^k\right| + \left|C_j^k\right| + \left|C_l^k\right|\right)\left(\left|C_j^k\right| + \left|C_i^k\right|\right)}\right) \sum_{t=1}^{n_{\mathrm{T}}} \overline{C_i^k}(t)^2 \\
& + \left(\frac{\left|C_j^k\right|^2 \left|C_l^k\right|}{\left(\left|C_i^k\right| + \left|C_j^k\right| + \left|C_l^k\right|\right)\left(\left|C_j^k\right| + \left|C_i^k\right|\right)}\right) \sum_{t=1}^{n_{\mathrm{T}}} \overline{C_j^k}(t)^2 \\
& + \frac{\left(\left|C_i^k\right| + \left|C_j^k\right|\right)\left|C_l^k\right|}{\left|C_i^k\right| + \left|C_j^k\right| + \left|C_l^k\right|} \sum_{t=1}^{n_{\mathrm{T}}} \overline{C_l^k}(t)^2 - 2\frac{\left|C_i^k\right|\left|C_l^k\right|}{\left|C_i^k\right| + \left|C_j^k\right| + \left|C_l^k\right|} \sum_{t=1}^{n_{\mathrm{T}}} \overline{C_i^k}(t)\overline{C_l^k}(t) \\
& - 2\frac{\left|C_j^k\right|\left|C_l^k\right|}{\left|C_i^k\right| + \left|C_j^k\right| + \left|C_l^k\right|} \sum_{t=1}^{n_{\mathrm{T}}} \overline{C_j^k}(t)\overline{C_l^k}(t) \\
& + 2\frac{\left|C_l^k\right|\left|C_i^k\right|\left|C_j^k\right|}{\left(\left|C_i^k\right| + \left|C_j^k\right| + \left|C_l^k\right|\right)\left(\left|C_j^k\right| + \left|C_i^k\right|\right)} \sum_{t=1}^{n_{\mathrm{T}}} \overline{C_i^k}(t)\overline{C_j^k}(t)
\end{aligned}
\tag{B.4}
$$

Développons maintenant la formule B.1 pour calculer la distance entre le cluster $C_i^k \cup C_j^k$ et le cluster C_l^k :

$$
\begin{aligned}
d\left(C_{i\cup j}^{k+1}, C_l^{k+1}\right) &= \frac{\left|C_i^k \cup C_j^k\right| + \left|C_l^k\right|}{\left|C_i^k \cup C_j^k\right| + \left|C_l^k\right|} \left\|\overline{C_i^k \cup C_j^k} - \overline{C_l^k}\right\|_2^2 \\
&= \frac{\left(\left|C_i^k\right| + \left|C_j^k\right|\right)\left|C_l^k\right|}{\left|C_i^k\right| + \left|C_j^k\right| + \left|C_l^k\right|} \left\| \frac{1}{\left|C_i^k\right| + \left|C_j^k\right|} \left(\left|C_i^k\right| \overline{C_i^k} + \left|C_j^k\right| \overline{C_j^k}\right) - \overline{C_l^k} \right\|_2^2
\end{aligned}
\tag{B.5}
$$

La dernière ligne de B.5 est obtenu en revenant à la définition du centroïde où on note $\boldsymbol{x}_j$ la série temporelle d'un élément j du cluster

$$
\begin{aligned}
\overline{C_i^k \cup C_j^k} &= \frac{1}{\left|C_i^k \cup C_j^k\right|} \sum_{v \in C_i^k \cup C_j^k} \boldsymbol{x}_v = \frac{1}{|C_i \cup C_j|} \left(\sum_{v \in C_i^k} \boldsymbol{x}_v + \sum_{v \in C_j^k} \boldsymbol{x}_v \right) \\
&= \frac{1}{\left|C_i^k\right| + \left|C_j^k\right|} \left(\left|C_i^k\right| \overline{C_i^k} + \left|C_j^k\right| \overline{C_j^k}\right)
\end{aligned}
$$

En développant B.5 en fonction de t, on a donc :

$$
\begin{aligned}
d\left(C_{i\cup j}^{k+1}, C_l^{k+1}\right) = & \frac{\left(\left|C_i^k\right| + \left|C_j^k\right|\right)\left|C_l^k\right|}{\left|C_i^k\right| + \left|C_j^k\right| + \left|C_l^k\right|} \left(\frac{1}{\left|C_i^k\right| + \left|C_j^k\right|} \left|C_i^k\right| \sum_{t=1}^{n_\mathrm{T}} \overline{C_i^k}(t) + \left|C_j^k\right| \sum_{t=1}^{n_\mathrm{T}} \overline{C_j^k}(t) \right)^2 \\
& - 2 \frac{\left(\left|C_i^k\right| + \left|C_j^k\right|\right)\left|C_l^k\right|}{\left(\left|C_i^k\right| + \left|C_j^k\right|\right)\left(\left|C_i^k\right| + \left|C_j^k\right| + \left|C_l^k\right|\right)} \left(\left|C_i^k\right| \sum_{t=1}^{n_\mathrm{T}} \overline{C_i^k}(t) + \left|C_j^k\right| \overline{C_j^k} \right) \sum_{t=1}^{n_\mathrm{T}} \overline{C_l^k}(t) \\
& + \frac{\left(\left|C_i^k\right| + \left|C_j^k\right|\right)\left|C_l^k\right|}{\left|C_i^k\right| + \left|C_j^k\right| + \left|C_l^k\right|} \sum_{t=1}^{n_\mathrm{T}} \overline{C_l^k}(t)^2 \\
& = \frac{\left|C_i^k\right|^2 \left|C_l^k\right|}{\left(\left|C_i^k\right| + \left|C_j^k\right|\right)\left(\left|C_i^k\right| + \left|C_j^k\right| + \left|C_l^k\right|\right)} \sum_{t=1}^{n_\mathrm{T}} \overline{C_i^k}(t)^2 \\
& + \frac{\left|C_j^k\right|^2 \left|C_l^k\right|}{\left(\left|C_i^k\right| + \left|C_j^k\right|\right)\left(\left|C_i^k\right| + \left|C_j^k\right| + \left|C_l^k\right|\right)} \sum_{t=1}^{n_\mathrm{T}} \overline{C_j^k}^2(t) \\
& + \frac{\left(\left|C_i^k\right| + \left|C_j^k\right|\right)\left|C_l^k\right|}{\left|C_i^k\right| + \left|C_j^k\right| + \left|C_l^k\right|} \sum_{t=1}^{n_\mathrm{T}} \overline{C_l^k}(t)^2 + 2 \frac{\left|C_i^k\right|\left|C_j^k\right|\left|C_l^k\right|}{\left(\left|C_i^k\right| + \left|C_j^k\right|\right)\left(\left|C_i^k\right| + \left|C_j^k\right| + \left|C_l^k\right|\right)} \sum_{t=1}^{n_\mathrm{T}} \overline{C_i^k}(t)\overline{C_j^k}(t) \\
& - 2 \frac{\left|C_i^k\right|\left|C_l^k\right|}{\left|C_i^k\right| + \left|C_j^k\right| + \left|C_l^k\right|} \sum_{t=1}^{n_\mathrm{T}} \overline{C_i^k}(t)\overline{C_l^k}(t) - 2 \frac{\left|C_j^k\right|\left|C_l^k\right|}{\left|C_i^k\right| + \left|C_j^k\right| + \left|C_l^k\right|} \sum_{t=1}^{n_\mathrm{T}} \overline{C_j^k}(t)\overline{C_l^k}(t)
\end{aligned}
\tag{B.6}
$$

Les dernières équations en B.4 et B.6 sont égales ce qui démontre la formule de Lance-Williams.

Annexe C

Illustration du calcul de distance en utilisant des tuiles rectangulaires

Cette annexe détaille le calcul de la matrice de distance en découpant la matrice d'entrée $\boldsymbol{X}$ en tuiles rectangulaires étape par étape. Considérons $\boldsymbol{X}$ une matrice de taille 6×11. Les différentes étapes de l'algorithme sont illustrées de la figure C.1 à la figure C.10 lorsqu'on considère une tuile rectangulaire qui couvre toutes les lignes mais seulement 3 colonnes. Sur les figures, les cases du quadrillage représentent un élément de la matrice, les rectangles(trait noir épais) délimitent les différentes tuiles de la matrice d'entrée. Les tuiles colorées montrent les tuiles utilisées à chaque étape pour la matrice d'entrée $\boldsymbol{X}$. Les tuiles carrées colorées dans la matrice de distance montre quelles parties de la matrice de distances est remplie.

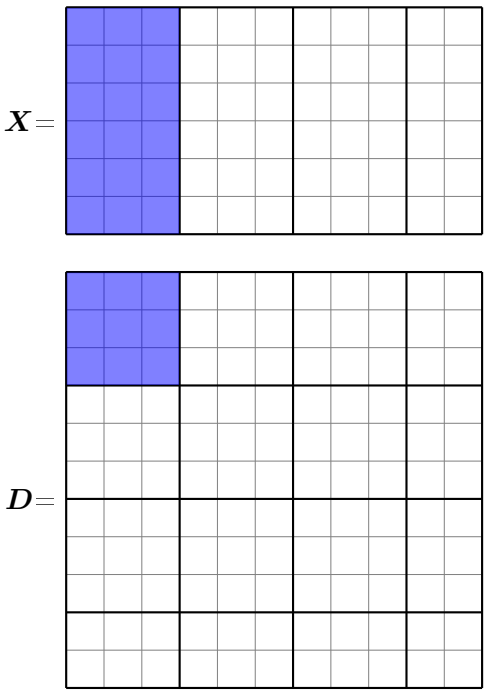


FIGURE C.1 – Étape 1.

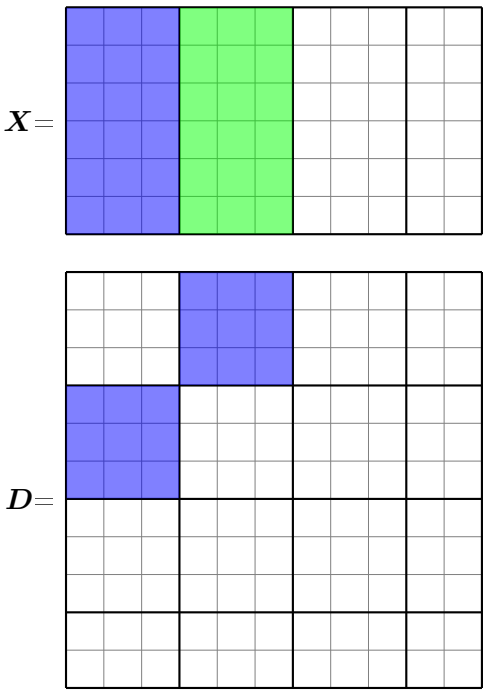


FIGURE C.2 – Étape 2.

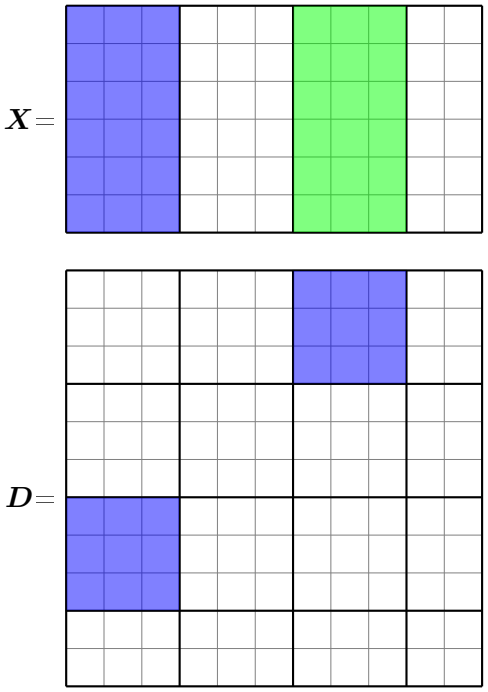


FIGURE C.3 – Étape 3.

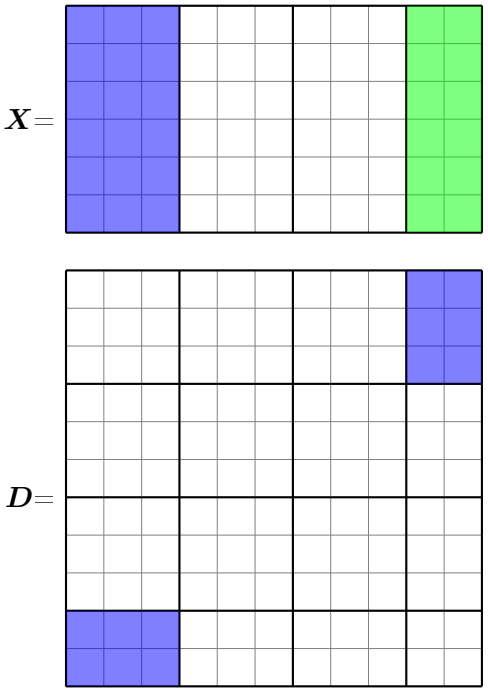


FIGURE C.4 – Étape 4.

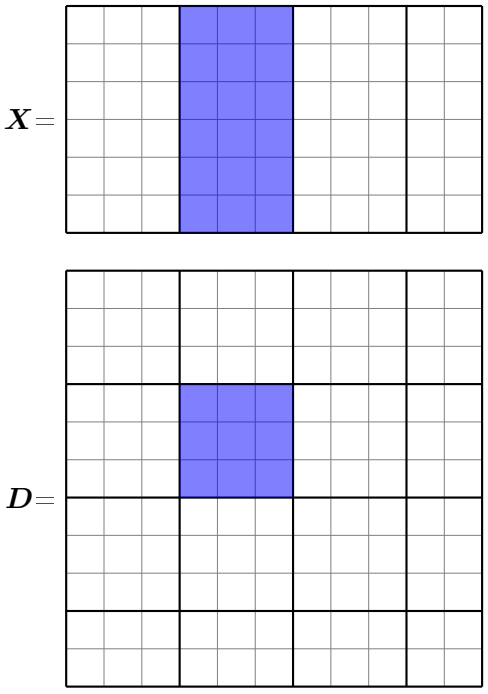


FIGURE C.5 – Étape 5.

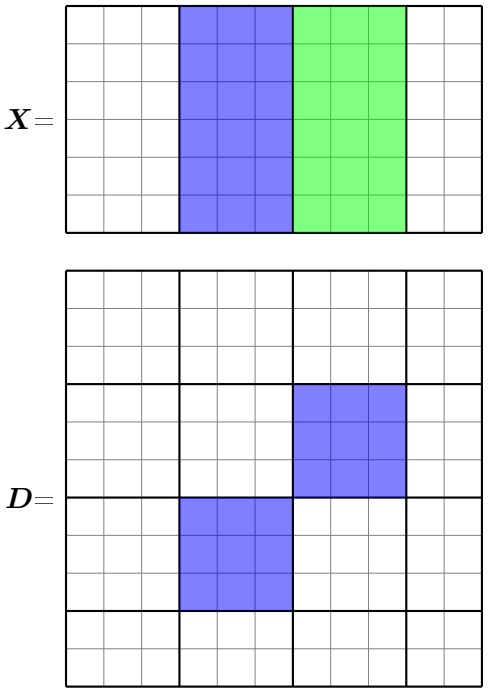


FIGURE C.6 – Étape 6.

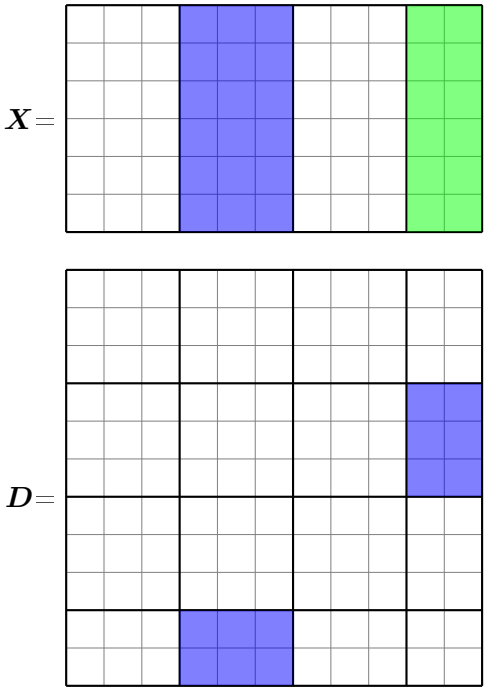


FIGURE C.7 – Étape 7.

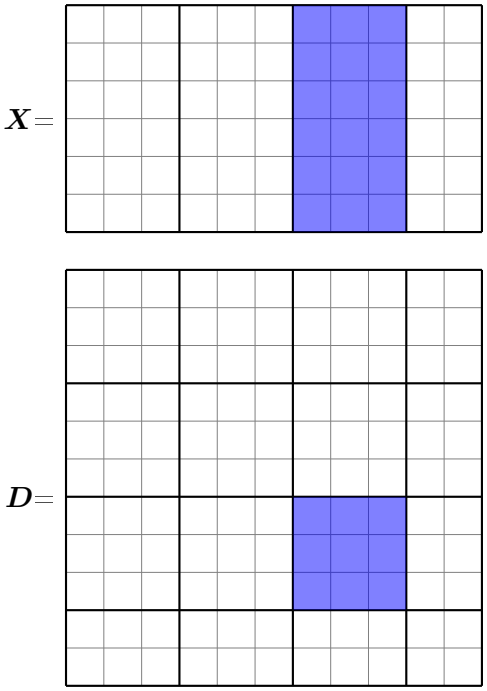


FIGURE C.8 – Étape 8.

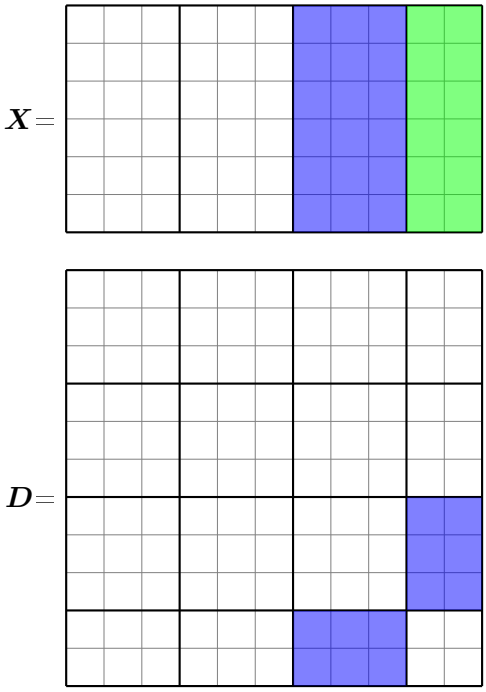


FIGURE C.9 – Étape 9.

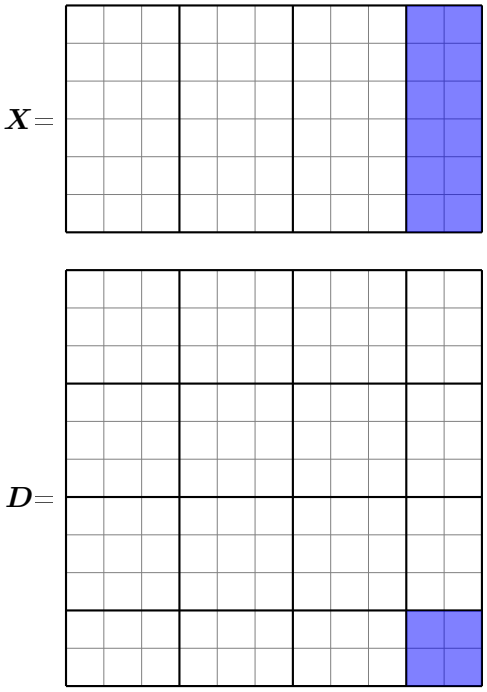


FIGURE C.10 – Étape 10.

Annexe D

Algorithme de Ward

Algorithme 17 : Algorithme de Ward complet

Entrée : $\boldsymbol{X}$ de taille $n_{\mathrm{T}} \times n_{\mathrm{T}}$; m_R ; m_C
Sortie : ***mergedClusterd*** matrice de taille $2 \times (n_{\mathrm{V}} - 1)$; ***parent*** vecteur de taille $2n_{\mathrm{V}} - 1$

`// Initialisation de` ***dimClusters*** `représentant le cardinal des parcelles`
dimClusters=1
`// Initialisation de` ***parent***
pour *i de 1 à* $2n_{\mathrm{V}} - 1$ **faire**
  parent(i)=i
`// Calcul de la distance et de` ***minDist***
Voir algorithme 13
`// Initialisation de` ***minDist***
pour *j de 1 à* n_{V} **faire**
  minDist(j)=$\min_{i\in\{1,\dots,n_{\mathrm{V}}\}} d_{ij}$; ***minLoc***(j)=$\arg\min_{i\in\{1,\dots,n_{\mathrm{V}}\}} d_{ij}$
pour *iter de 1 à* n_{V}*-1* **faire**
  `// Trouver quelles parcelles fusionner`
  i=$arg\,min_{j\in\{1,\dots,n_{\mathrm{V}}\}}$***minDist***$(j)$; j=***minLoc***(i)
  `// La nouvelle fusion est placée à l'indice min(i,j)`
  upInd=$\min(i,j)$; delInd= $\max(i,j)$
  `// Garder les dimensions des parcelles upInd et delInd`
  delDim← ***dimClusters***(delInd) ; upDim← ***dimClusters***(upInd)
  `// Mise à jour de` ***dimClusters***
  dimClusters(upInd)←upDim+delDim
  dimClusters(delInd)←0
  `// Mise à jour de la matrice de distance`
  pour *i de 1 à upInd-1* **faire**
    $\alpha \leftarrow \frac{upDim+\boldsymbol{dimClusters}(i)}{upDim+delInd+\boldsymbol{dimClusters}(i)}$; $\beta \leftarrow \frac{delDim+\boldsymbol{dimClusters}(i)}{upDim+delInd+\boldsymbol{dimClusters}(i)}$; $\gamma \leftarrow \frac{\boldsymbol{dimClusters}(i)}{upDim+delInd+\boldsymbol{dimClusters}(i)}$
    $\boldsymbol{D}(i, upInd) \leftarrow \alpha\boldsymbol{D}(i, upInd) + \beta\boldsymbol{D}(i, delkInd) + \gamma\boldsymbol{D}(delInd, upInd)$
  pour *i de upInd+1 à* n_{V} **faire**
    $\alpha \leftarrow \frac{upDim+\boldsymbol{dimClusters}(i)}{upDim+delInd+\boldsymbol{dimClusters}(i)}$; $\beta \leftarrow \frac{delDim+\boldsymbol{dimClusters}(i)}{upDim+delInd+\boldsymbol{dimClusters}(i)}$; $\gamma \leftarrow \frac{\boldsymbol{dimClusters}(i)}{upDim+delInd+\boldsymbol{dimClusters}(i)}$
    $\boldsymbol{D}(i, upInd) \leftarrow \alpha\boldsymbol{D}(i, upInd) + \beta\boldsymbol{D}(i, delkInd) + \gamma\boldsymbol{D}(delInd, upInd)$
  pour *i de 1 à* n_{V} **faire**
    D(i,delInd)←0
  `// Mise à jour de` ***minDist*** `et` ***minLoc***
  Voir algorithme 6
  `// Remplir` ***mergedClusters*** `et` ***parent***
  i←upInd
  tant que ***parent****(i)*$\neq i$ **faire**
    i← ***parent***(i)
  j←delInd
  tant que ***parent****(j)*$\neq j$ **faire**
    i← ***parent***(j)
  mergedClusters(:,iter)← (i,j)
  parent(i)← n_{V}+iter ; ***parent***(j)← n_{V}+iter

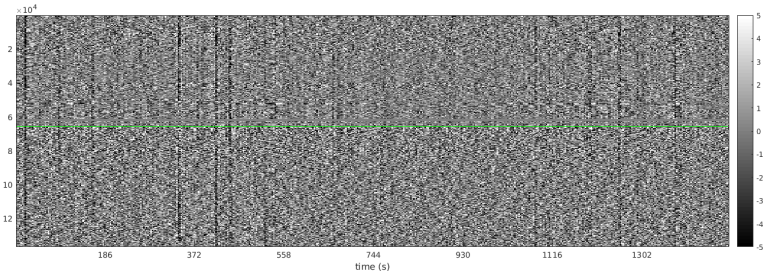
Annexe E

Données supplémentaires

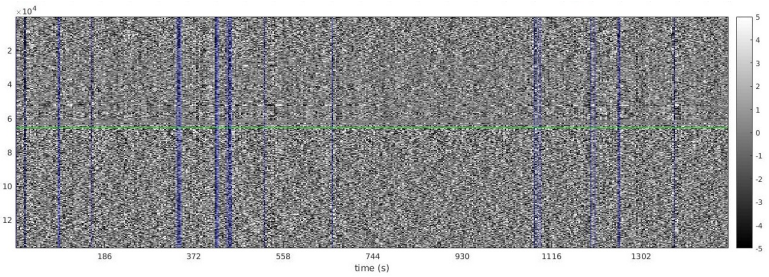
Cette annexe regroupe les données supplémentaires non présentées dans le manuscrit.

E.1 Effet de la censure sur les diagrammes de Power

Cette section présente l'effet de la censure par la méthode de l'ellipse de confiance avec une probabilité de 95% pour les sujets et sessions non présentés dans le chapitre 5.



(a) Original

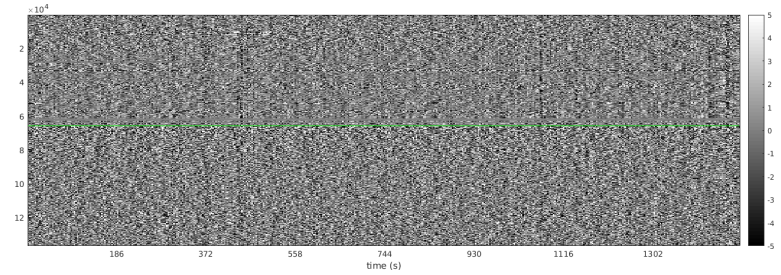


(b) Après censure

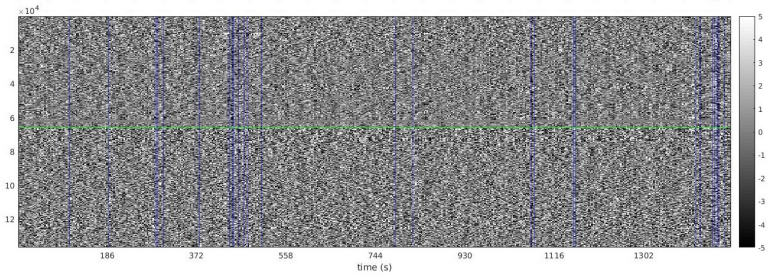
FIGURE E.1 – Diagramme de Power avant/après censure pour la session 1 du sujet n^o 1.

E.2 Analyse intra-individuelle du signal de pression

Nous présentons ici la caractérisation des stimulations de déglutition analysées en section 9.1 pour les quatre sujets. Pour chaque stimulation, sont donnés le nombre de pics au-dessus de 10% avec entre chaque pic un écart minimum de trois secondes, l'amplitude de la variation du signal en pourcentage, sa localisation dans le paradigme en secondes ainsi que son retard ou son avance par rapport à l'ordre de déglutition.

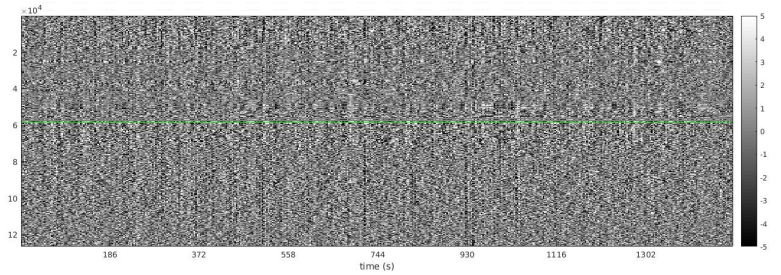


(a) Original

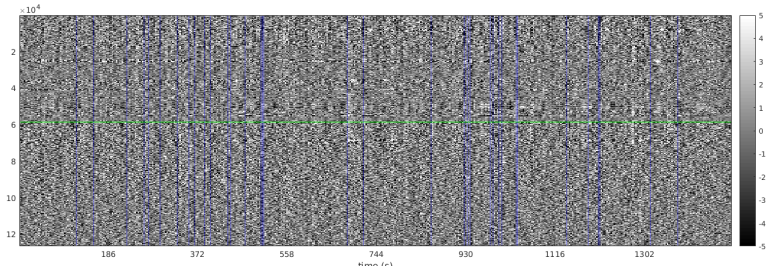


(b) Après censure

FIGURE E.2 – Diagramme de Power avant/après censure pour la session 2 du sujet n^o 1.

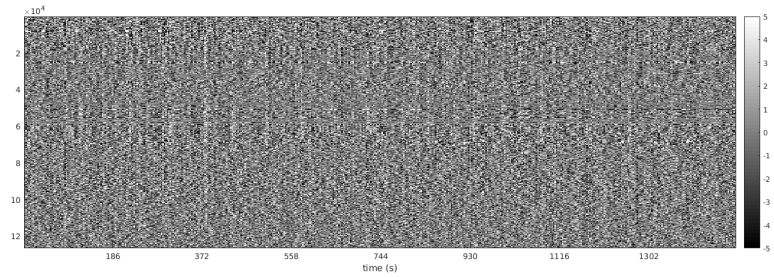


(a) Original

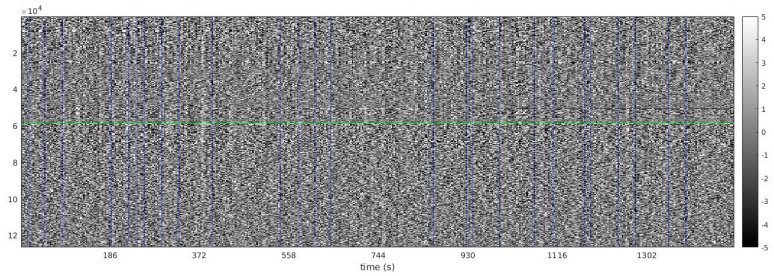


(b) Après censure

FIGURE E.3 – Diagramme de Power avant/après censure pour la session 1 du sujet n^o 2.

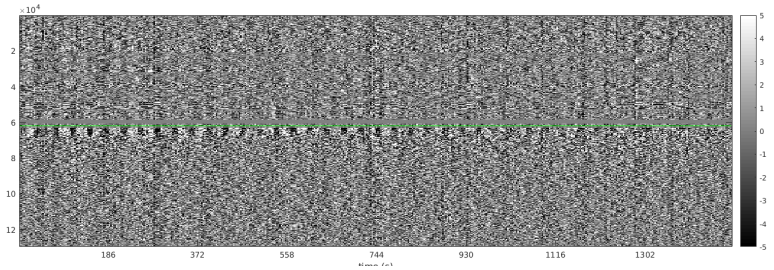


(a) Original

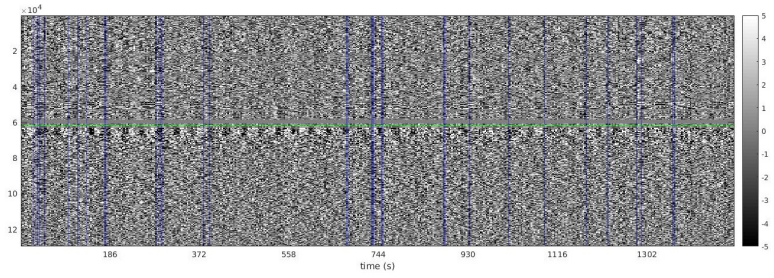


(b) Après censure

FIGURE E.4 – Diagramme de Power avant/après censure pour la session 2 du sujet n^o 2.

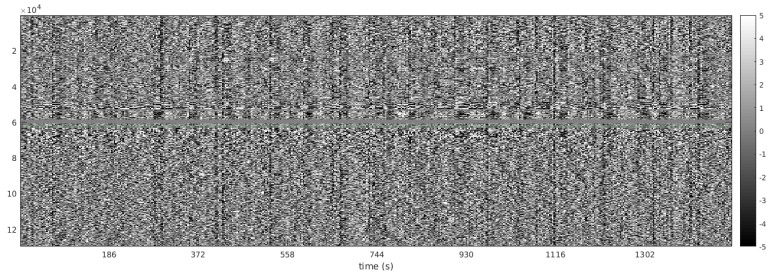


(a) Original

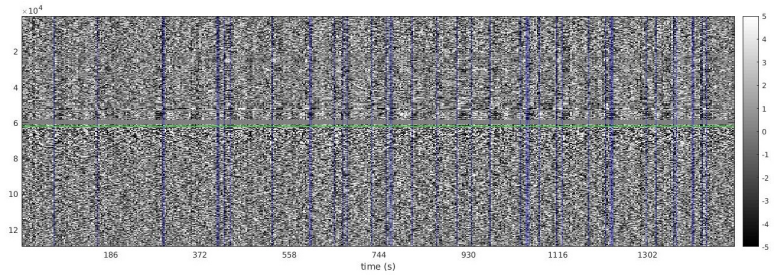


(b) Après censure

FIGURE E.5 – Diagramme de Power avant/après censure pour la session 1 du sujet n^o 3.

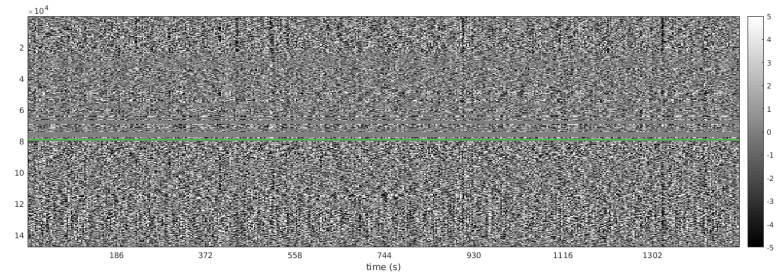


(a) Original

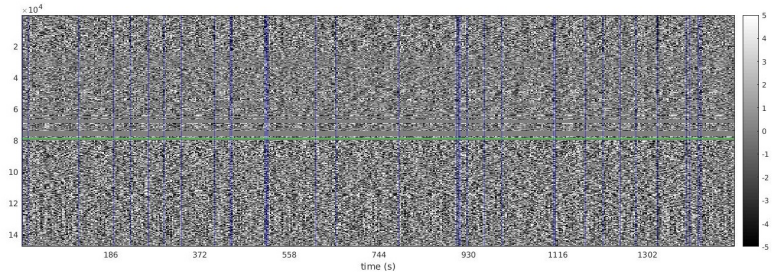


(b) Après censure

FIGURE E.6 – Diagramme de Power avant/après censure pour la session 2 du sujet n^o 3.



(a) Original



(b) Après censure

FIGURE E.7 – Diagramme de Power avant/après censure pour la session 2 du sujet n^o 4.

E.2.1 Sujet 1

TABLE E.1 – Caractérisation des stimulations selon leur nombre de pics pour le sujet 1.

N^o stimulation	**Nombre de pics**	**Amplitude des pics (en %)**	**Localisation des pics (en s)**	**Retard par rapport à l'ordre de déglutition (en s)**
1	4	42.01 33.71 37.71 35.75	13.48 18.11 23.91 27.07	1.44 6.08 11.88 15.03
2	4	41.12 38.63 18.86 27.04	13.60 20.87 27.03 33.11	1.55 8.83 14.99 21.07
3	3	38.37 46.19 11.00	13.88 21.39 33.39	1.84 9.36 21.36
4	4	42.26 44.62 23.68 11.35	13.80 21.23 30.75 33.83	1.76 9.20 18.71 21.79
5	3	42.48 44.37 39.20	13.44 22.39 25.75	1.40 10.36 13.72
6	2	38.65 43.32	14.04 29.35	1.99 17.31
7	3	37.51 38.30 34.57	13.92 19.71 28.55	1.88 7.68 16.52
8	3	40.28 42.31 36.36	14.40 20.39 28.31	2.35 8.35 16.27
9	3	39.77 37.84 37.56	13.88 17.59 23.55	1.84 5.56 11.52
10	3	41.71 40.91 35.05	13.56 17.99 28.63	1.52 5.96 16.60
11	3	39.89 38.24 36.81	13.76 18.51 24.47	1.73 6.48 12.44
12	3	41.00 38.11 39.17	13.84 20.23 33.67	1.79 8.19 21.63
13	3	46.09 44.27 36.55	14.12 20.47 30.51	2.08 8.44 18.48
14	3	43.13 42.35 36.12	13.68 20.03 31.99	1.65 8.01 19.96
15	4	44.69 38.69 12.52 35.11	13.92 18.87 26.67 31.91	1.88 6.84 14.64 19.88

Suite à la page suivante

TABLE E.1 – *Suite de la page précédente*

N^o stimulation	**Nombre de pics**	**Amplitude des pics (en %)**	**Localisation des pics (en s)**	**Retard par rapport à l'ordre de déglutition (en s)**
16	3	42.47 45.51 12.24	13.76 23.55 31.27	1.73 11.52 19.24
17	3	40.50 40.29 36.63	13.72 18.75 31.75	1.67 6.71 19.71
18	2	44.77 11.46	13.84 29.83	1.80 17.80
19	4	17.64 44.53 44.75 43.67	0.16 13.32 18.23 30.31	-11.88 1.28 6.20 18.28
20	2	37.72 33.27	14.16 35.31	2.12 23.28
21	4	44.97 47.06 11.66 43.55	14.08 22.43 27.03 30.63	2.03 10.39 14.99 18.59
22	2	47.18 46.46	14.32 22.11	2.28 10.08
23	3	42.27 36.57 38.24	13.56 19.47 30.63	1.51 7.43 18.59
24	2	44.33 45.40	13.80 23.15	1.74 11.10
25	3	39.28 35.85 32.02	14.04 20.15 31.67	2.00 8.12 19.64
26	2	47.36 44.02	14.20 25.95	2.16 13.92
27	2	42.10 39.76	13.96 19.79	1.92 7.76
28	3	39.11 11.95 36.22	13.68 18.07 21.95	1.62 6.02 9.89
29	2	41.32 40.75	14.20 20.43	2.17 8.41
30	2	41.49 46.50	13.76 30.67	1.71 18.63
31	4	43.81 12.05 40.05 37.20	14.04 18.59 21.63 31.15	2.00 6.56 9.60 19.11
32	2	43.38 39.02	14.16 28.91	2.13 16.88
33	3	40.94 41.29 10.58	14.28 22.19 32.63	2.23 10.15 20.59
34	2	44.98 40.33	13.84 26.51	1.80 14.48

Suite à la page suivante

TABLE E.1 – *Suite de la page précédente*

N^o stimulation	**Nombre de pics**	**Amplitude des pics (en %)**	**Localisation des pics (en s)**	**Retard par rapport à l'ordre de déglutition (en s)**
35	3	40.94 43.00 45.27	13.56 21.15 33.99	1.53 9.12 21.96
36	2	49.17 35.84	13.88 30.99	1.83 18.95
37	4	50.56 11.53 52.46 11.12	14.32 19.31 23.23 33.19	2.28 7.28 11.20 21.16
38	2	42.41 39.83	13.72 24.11	1.69 12.08
39	2	44.79 44.79	14.00 23.03	1.94 10.98
40	2	46.42 40.61	13.44 23.03	1.39 10.99
41	3	50.15 48.79 44.35	13.80 18.31 23.43	1.77 6.29 11.41
42	2	44.15 43.87	13.72 21.79	3.69 11.77
43	3	43.31 43.31 40.51	13.92 19.87 28.95	3.90 9.86 18.94
44	2	43.66 38.18	13.88 29.83	1.85 17.81
45	2	44.57 41.67	14.12 22.35	2.10 10.33
46	3	47.71 23.76 41.44	13.20 22.59 27.47	3.17 12.57 17.45
47	2	42.37 43.95	14.16 19.47	2.13 7.45
48	2	45.14 44.64	13.68 29.79	2.66 18.77
49	2	47.25 47.25	13.72 19.67	3.70 9.66
50	2	43.77 42.93	14.04 25.79	1.99 13.75
51	2	47.54 47.54	13.68 19.35	1.64 7.32
52	2	43.47 43.47	13.48 24.83	3.43 14.79
53	3	47.68 38.95 46.02	13.80 18.43 28.19	1.75 6.39 16.15
54	2	43.21 43.21	14.04 28.71	3.00 17.67
55	2	44.08 41.68	13.56 18.55	0.50 5.50
56	2	45.31 45.31	13.36 24.39	1.30 12.34

Suite à la page suivante

TABLE E.1 – *Suite de la page précédente*

N^o stimulation	**Nombre de pics**	**Amplitude des pics (en %)**	**Localisation des pics (en s)**	**Retard par rapport à l'ordre de déglutition (en s)**
57	3	42.35 42.35 42.35	13.92 21.59 32.27	2.86 10.54 21.22
58	2	46.59 45.09	13.68 23.47	3.63 13.43
59	2	43.90 38.34	14.04 20.39	2.00 8.35
60	2	48.84 48.47	13.48 17.87	2.43 6.83
61	3	43.53 40.17 39.88	13.84 21.15 28.51	2.78 10.10 17.46
62	4	44.14 13.23 10.41 35.69	13.92 21.43 27.75 33.59	1.88 9.39 15.71 21.55
63	3	49.52 47.75 47.60	14.04 20.87 30.23	3.99 10.83 20.19
64	2	43.26 39.39	14.04 19.31	1.99 7.27
65	2	42.37 38.25	14.12 21.99	2.10 9.97
66	1	41.41	13.76	1.73
67	3	45.06 44.85 16.12	13.60 25.63 34.55	1.58 13.61 22.53
68	4	37.74 45.31 40.33 36.37	1.56 13.76 20.43 33.03	-10.47 1.72 8.40 21.00
69	3	44.94 39.20 39.06	14.64 19.31 30.87	2.64 7.31 18.87
70	2	46.56 39.54	13.56 30.27	3.55 20.26
71	3	43.55 34.51 17.30	13.76 19.83 33.71	1.75 7.83 21.70
72	3	39.45 46.31 43.38	0.52 13.72 25.03	-10.49 2.71 14.03
73	3	45.14 33.86 38.54	14.24 20.55 31.27	4.23 10.55 21.26
74	2	44.55 34.23	13.84 24.11	1.83 12.11
75	2	44.34 34.08	13.76 21.47	1.75 9.47
76	2	42.25 36.77	13.44 21.91	1.42 9.90

Suite à la page suivante

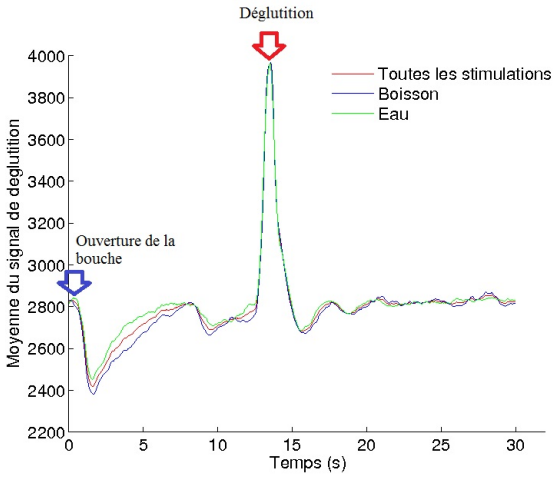


FIGURE E.8 – Moyenne des signaux de déglutition sur toutes les stimulations pour le sujet 2.

TABLE E.1 – *Suite de la page précédente*

N^o stimulation	**Nombre de pics**	**Amplitude des pics (en %)**	**Localisation des pics (en s)**	**Retard par rapport à l'ordre de déglutition (en s)**
77	3	41.22 38.33 37.55	13.84 23.23 33.59	1.83 11.23 21.58
78	1	40.67	13.84	0.82
79	4	43.36 48.37 16.85 38.64	0.16 13.80 19.11 22.31	-12.86 0.78 6.10 9.29
80	4	45.06 42.87 16.75 34.41	1.28 13.88 17.83 21.11	-10.72 1.88 5.84 9.12

E.2.2 Sujet 2

La figure E.8 représente la moyenne du signal de déglutition sur toutes les stimulations, sur les stimulations avec boisson sucrée et sur les stimulations avec eau pure. On constate qu'il y a peu de différences entre les deux conditions. On remarque, comme pour le sujet 1, un léger pic au début correspondant à l'ouverture de la bouche par le sujet et un pic plus élevé représentant la déglutition du sujet. Le tableau E.2 montre les pics trouvés sur l'ensemble des stimulations. Les pics n^{os} 2, 4 et 5 ayant un écart-type de zéro, ce sont des pics trouvés pour une seule stimulation. Le tableau E.5 montre les caractéristiques des pics pour chaque stimulation. En le rapprochant du tableau E.2, on y constate que les différents pics sont répartis entre les différentes stimulations. En effet, le pic n^o 1 est présent pour les stimulations 8 et 57, le pic n^{o}2 pour la stimulation 6, le pic n^{o}4 pour la stimulation 2, le pic 5 est présent pour les stimulations 2, 39 41, le pic n^{o}6 pour les stimulations 7 et 31, le pic n^{o}7 pour les stimulations 10, 23 et 47 et le pic n^o 8 pour les stimulations 28 et 56. Seul le pic n^{o}3 qui correspond au pic après l'ordre de déglutition se retrouve pour toutes les stimulations.

La *slope statistic* (**équation 6.7**) détermine pour ce sujet-ci le nombre optimal de parcelles à deux. La figure E.9 montre les deux centroïdes. On remarque qu'ils se ressemblent beaucoup, différant uniquement par l'amplitude du pic négatif situé dans la phase d'avertissement, qui pourrait correspondre à l'ouverture de la bouche. Le tableau E.3 caractérise l'unique pic (positif) des deux centroïdes, qui correspond au pic de déglutition. Le tableau E.4 montre que les stimulations

N^o de pic	**Localisation (en s)**	**Amplitude du pic (en %)**
1	0.26±0.31	31.09± 0.69
2	3.96±0.00	37.88±0.00
3	13.40±0.20	43.71±3.63
4	17.07±0.00	10.00±0.00
5	20.91± 0.00	40.36±0.00
6	27.70 ±0.64	36.35±3.01
7	30.79±1.34	36.65±6.08
8	35.21±0.42	30.66±2.26

TABLE E.2 – Caractérisation des pics sur l'ensemble des stimulations pour le sujet 2.

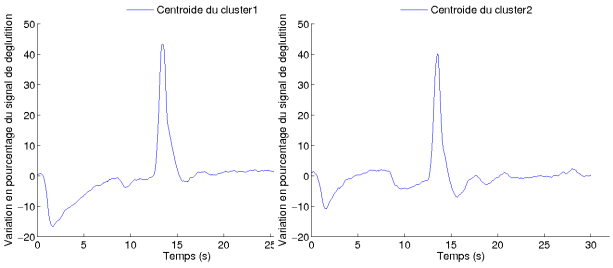


FIGURE E.9 – Centroides des parcelles pour le sujet 2.

se sont réparties à part égale dans les deux parcelles. On y remarque que les stimulations qui présentent des pics en tout début du paradigme sont regroupées dans la parcelle 2.

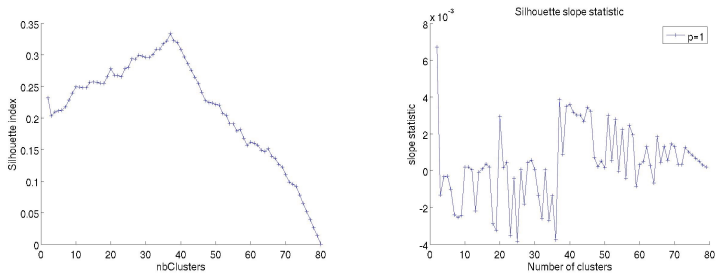
N^o de pic	**Localisation (en s)**	**Amplitude du pic (en %)**
1	13.46±0.08	41.87±2.24

TABLE E.3 – Caractérisation des pics sur l'ensemble des centroïdes pour le sujet 2.

N^o de la parcelle	**N^{os} des stimulations dans la parcelle**
1	1,2,3,4,5,7,9,11,12,13,14,16,17,19,27,29,33 36,40,41,43,44,46,47,48,51,52,56,59,60,64 67,68,69,70,72,73,75,78,79
2	6,8,10,15,18,20,21,22,23,24,25,26,28,30,31 32,34,35,37,38,39,42,45,49,50,53,54,55,57 58,61,62,63,65,66,71,74,76,77,80

TABLE E.4 – Répartition des stimulations dans les parcelles pour le sujet 2.

On peut voir sur la figure E.10 que l'indice de silhouette est plus élevé lorsqu'on augmente le nombre de parcelles mais on n'y observe pas de saut important entre deux nombres de parcelles successifs. Cependant, la *slope statistic* montre que le deuxième choix pour le nombre optimal de parcelles est 37. Ce deuxième choix suggère une plus grande variabilité intra-individuelle que le nombre optimal de parcelles fixé à deux. Un tel choix mettrait peut-être en évidence des centroïdes avec deux pics rapprochés et correspondrait plus à la répartition des différents pics entre les parcelles.



(a) Indice de silhouette (b) *Slope statistic*

FIGURE E.10 – Détermination du nombre optimal de parcelles pour le sujet 2.

TABLE E.5 – Caractérisation des stimulations selon leur nombre de pics pour le sujet 2.

N^o stimulation	**Nombre de pics**	**Amplitude des pics (en %)**	**Localisation des pics (en s)**	**Retard par rapport à l'ordre de déglutition (en s)**
1	1	49.52	13.40	1.36
2	2	48.33 10.00	13.36 17.07	1.33 5.04
3	1	48.02	13.32	1.29
4	1	47.22	13.32	1.28
5	1	43.83	13.08	1.03
6	2	37.88 44.07	3.96 13.60	-8.08 1.56
7	1	48.09	13.24	1.21
8	2	31.58 48.89	0.48 13.64	-11.55 1.60
9	1	45.45	13.24	1.21
10	2	44.28 31.71	13.52 31.47	1.46 19.41
11	1	45.91	13.20	1.17
12	1	44.29	13.20	1.14
13	1	48.15	13.44	1.38
14	1	44.70	13.28	1.23
15	1	46.06	13.68	1.63
16	1	43.61	13.28	1.23
17	1	44.42	13.16	1.13
18	1	44.53	13.64	1.59
19	1	44.65	13.28	1.23
20	1	43.54	13.16	1.11
21	1	45.70	13.16	1.12
22	1	43.83	13.44	1.41
23	2	42.92 32.60	13.20 31.11	1.16 19.07
24	1	38.12	13.52	1.49
25	1	44.77	13.32	1.28
26	1	41.74	13.60	1.54
27	2	45.16 32.33	13.16 29.75	1.12 17.72
28	2	43.55 32.26	13.56 35.51	1.52 23.48
29	1	46.31	13.20	1.15
30	1	40.60	13.48	1.45

Suite à la page suivante

TABLE E.5 – *Suite de la page précédente*

N^o stimulation	**Nombre de pics**	**Amplitude des pics (en %)**	**Localisation des pics (en s)**	**Retard par rapport à l'ordre de déglutition (en s)**
31	2	35.56 39.81	13.36 26.99	1.32 14.96
32	1	43.79	13.64	1.61
33	1	44.05	14.00	1.95
34	1	31.80	13.52	1.48
35	2	45.05 36.90	13.16 30.31	1.13 18.28
36	1	43.38	13.24	1.19
37	1	47.87	13.64	1.60
38	1	43.04	13.16	1.13
39	2	46.22 40.36	13.48 20.91	1.44 8.88
40	1	44.52	13.32	1.27
41	2	46.77 46.77	13.20 28.47	1.16 16.43
42	1	43.51	13.40	1.36
43	1	42.78	13.20	1.15
44	1	43.62	13.24	1.18
45	2	46.17 34.32	13.60 27.87	2.54 16.82
46	1	45.93	13.12	2.07
47	2	47.72 33.14	13.40 32.95	1.34 20.90
48	1	43.63	13.28	1.22
49	1	47.28	13.28	1.22
50	1	43.22	13.20	1.13
51	1	48.91	13.60	2.56
52	1	40.21	13.16	1.12
53	1	37.51	13.60	1.57
54	1	41.18	13.12	0.12
55	1	33.37	13.24	1.25
56	2	44.69 29.07	13.12 34.91	1.14 22.93
57	2	30.60 39.38	0.04 13.56	-11.93 1.59
58	1	44.15	13.84	1.85
59	1	39.24	13.64	1.66
60	1	43.26	13.48	1.53
61	2	31.52 34.23	13.44 30.59	0.50 17.66
62	1	37.54	13.72	1.78
63	1	40.27	13.60	2.66
64	1	43.01	13.44	3.51
65	2	41.06 34.92	13.80 28.23	1.86 16.29
66	1	39.04	13.56	3.62
67	1	46.65	13.56	3.64
68	1	45.89	13.52	1.59
69	1	49.54	13.68	2.76
70	1	45.99	13.04	3.13
71	1	43.57	13.52	1.59

Suite à la page suivante

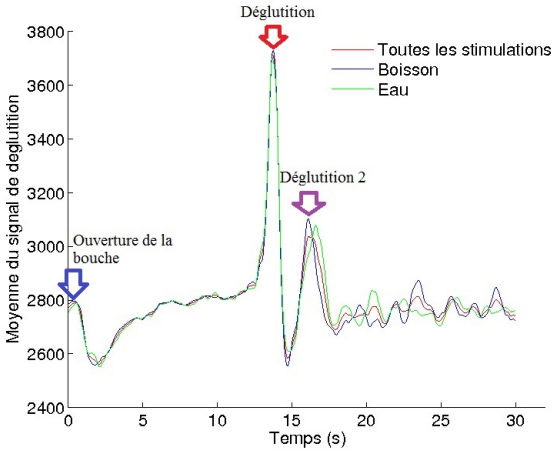


FIGURE E.11 – Moyennes des signaux de déglutition pour toutes les stimulations pour le sujet 3.

TABLE E.5 – *Suite de la page précédente*

N^o stimulation	**Nombre de pics**	**Amplitude des pics (en %)**	**Localisation des pics (en s)**	**Retard par rapport à l'ordre de déglutition (en s)**
72	1	45.13	13.28	1.36
73	2	42.89 45.54	13.52 31.63	1.58 19.70
74	1	42.33	13.36	1.43
75	1	45.89	13.28	1.34
76	1	44.03	13.52	3.58
77	1	44.18	13.60	1.66
78	1	45.69	13.36	2.42
79	1	41.96	13.28	1.34
80	1	42.53	13.60	1.65

E.2.3 Sujet 3

La figure E.11 montre la moyenne sur toutes les stimulations, sur les stimulations avec de la boisson sucrée et sur les stimulations avec de l'eau. On retrouve comme pour les deux premiers sujets un léger pic signalant l'ouverture de la bouche (flèche bleue) et un pic plus élevé correspondant à la déglutition (flèche rouge). Cependant, ce sujet se distingue par la présence d'un autre pic se situant environ 3 secondes après le pic de déglutition et d'amplitude moins élevé que le premier pic de déglutition (flèche violette). Ce pic peut s'analyser comme une deuxième déglutition du sujet.

Le tableau E.6 montre les différents pics trouvés sur l'ensemble des stimulations. On y trouve un pic pendant la phase d'avertissement qui peut correspondre à l'ouverture de la bouche. Le pic n^{o}2 a un écart-type de zéro donc n'est présent que pour une seule stimulation. Le pic n^{o}3 arrive peu après l'ordre de déglutition et correspond donc au pic de déglutition. Le pic n^o 4 se situe en fin de phase de déglutition et les pics suivants se situent durant la phase de repos. Ils montrent que la déglutition s'étale dans le temps. On note cependant que leurs amplitudes sont plus faibles que celle du premier pic de déglutition. La table E.9 montre le détail des pics et leurs caractéristiques pour chaque stimulation. On y voit la variabilité intra-individuelle : deux stimulations successives pouvant être très différentes (par exemple les stimulations n^{os} 5, 6 et 7).

On peut parcelliser ces stimulations en 30 parcelles selon la *slope statistic*. La table E.8 montre la répartition des stimulations dans les différentes parcelles. On remarque que les parcelles n^{os} 3, 9, 13, 15, 21 et 25 ne comportent qu'une seule stimulation.La parcelle n^o 26 contient des

N^o de pic	Localisation (en s)	Amplitude du pic (en %)
1	0.80±0.89	23.07±12.17
2	6.08±0.00	11.10±0.00
3	13.87±0.70	40.04±7.95
4	18.35±1.33	29.92±7.49
5	23.53±1.64	23.71±8.93
6	28.09±0.77	26.28±6.09
7	31.98±1.39	25.32±8.88

TABLE E.6 – Caractérisation des pics sur l'ensemble des stimulations pour le sujet 3.

stimulations avec plusieurs pics (excepté la stimulation 38). Les figures E.12 à E.16 montrent les centroïdes des différentes parcelles. Ces figures illustrent l'intra-variabilité de la déglutition pour le sujet 3. En effet, on y trouve des parcelles dont le signal n'est caractérisé que par deux pics comme les parcelles n^{os} 5, 7 et 26 ; et des parcelles avec un grand nombre de pics (par exemple la parcelle 16). Le tableau E.7 montre les pics trouvés sur l'ensemble des centroïdes. On y trouve le pic n^{o}1 pendant la phase d'avertissement correspondant à l'ouverture de la bouche, le pic n^{o}2 correspond au pic de déglutition puis suivent des pics de déglutitions en fin de phase de déglutition et pendant la phase de repos.

N^o de pic	Localisation (en s)	Amplitude du pic (en %)
1	1.09±1.35	20.03±5.54
2	13.87±0.75	39.63±6.93
3	17.81±0.88	26.47±10.03
4	21.27±0.84	22.98±5.87
5	24.22±1.03	21.46±8.53
6	28.25±1.08	$19.13_pm4.72$

TABLE E.7 – Caractérisation des pics sur l'ensemble des centroïdes pour le sujet 3.

La figure E.17(a) montre l'indice de silhouette (**cf. équation 6.2**) pour le sujet 3 : on voit que l'indice de silhouette diminue entre deux et quatre parcelles avec un saut plus important que par la suite, puis qu'il remonte jusqu'aux alentours de 30 parcelles pour enfin diminuer. On note également sur la figure E.17(b) que le deuxième maximum pour la *slope statistic* (**cf. équation 6.7**) correspond à une parcellisation en trois parcelles avec une *slope statitic* très proche de celle correspondant à 30 parcelles.

TABLE E.9 – Caractérisation des stimulations selon leur nombre de pics pour le sujet 3.

N^o stimulation	Nombre de pics	Amplitude des pics (en %)	Localisation des pics (en s)	Retard par rapport à l'ordre de déglutition (en s)
1	4	43.84 27.68 31.45 24.69	13.48 17.51 23.51 32.39	1.43 5.47 11.47 20.35
2	3	45.66 34.27 31.92	13.52 20.35 33.43	1.48 8.32 21.40
3	3	42.77 36.00 25.12	13.60 21.79 25.63	1.56 9.76 13.60

Suite à la page suivante

TABLE E.9 – *Suite de la page précédente*

N^o stimulation	**Nombre de pics**	**Amplitude des pics (en %)**	**Localisation des pics (en s)**	**Retard par rapport à l'ordre de déglutition (en s)**
4	2	41.49	13.60	1.55
		28.49	20.31	8.27
5	3	43.98	13.20	1.17
		29.35	17.35	5.32
		28.22	27.07	15.04
6	1	44.12	13.44	1.40
7	2	44.48	13.96	1.91
		30.79	21.47	9.43
8	4	43.30	13.64	1.60
		29.30	18.91	6.88
		28.81	22.27	10.24
		29.51	29.11	17.08
9	2	37.38	14.12	2.08
		29.72	19.39	7.36
10	2	42.37	13.52	1.47
		32.28	17.87	5.83
11	3	43.88	13.64	1.59
		27.65	19.63	7.59
		30.32	26.31	14.26
12	2	32.58	16.60	4.55
		15.09	31.59	19.55
13	4	43.73	13.68	1.64
		29.62	17.03	5.00
		34.81	23.35	11.32
		31.02	29.15	17.12
14	1	34.39	13.96	1.93
15	2	45.24	13.92	1.87
		33.29	21.75	9.71
16	2	31.51	13.60	1.56
		41.16	18.55	6.52
17	3	30.60	13.88	1.83
		32.49	16.95	4.91
		31.51	20.39	8.35
18	1	34.83	16.28	4.22
19	3	42.45	13.72	1.69
		30.24	16.79	4.76
		35.67	22.99	10.96
20	3	45.64	13.72	1.67
		15.69	25.15	13.11
		32.34	32.19	20.15
21	2	46.14	13.80	1.76
		41.70	29.59	17.56
22	2	43.21	13.56	1.52
		34.12	23.55	11.52
23	4	51.00	13.68	1.63
		37.80	18.19	6.15
		15.82	21.35	9.31
		14.19	24.39	12.35
24	3	41.14	1.68	-10.36
		30.28	13.88	1.84
		29.28	18.15	6.12

Suite à la page suivante

TABLE E.9 – *Suite de la page précédente*

N^o stimulation	**Nombre de pics**	**Amplitude des pics (en %)**	**Localisation des pics (en s)**	**Retard par rapport à l'ordre de déglutition (en s)**
25	2	40.20 28.46	13.76 21.99	1.73 9.96
26	2	36.50 13.66	14.08 22.39	2.03 10.35
27	2	45.49 40.21	13.60 25.39	1.56 13.36
28	3	28.44 28.80 28.23	14.12 18.43 22.15	2.08 6.40 10.12
29	3	11.10 44.83 24.21	6.08 13.76 31.95	-5.96 1.71 19.91
30	1	42.68	13.92	1.88
31	2	30.61 31.26	13.68 18.15	1.64 6.12
32	2	29.81 46.03	0.16 13.64	-11.88 1.59
33	4	39.35 12.86 10.26 26.60	16.52 20.99 25.31 28.63	4.48 8.96 13.28 16.60
34	2	43.46 31.02	14.24 20.63	2.21 8.60
35	3	46.76 13.59 21.40	13.32 23.71 28.59	1.27 11.67 16.55
36	3	30.95 36.00 10.15	13.88 17.23 23.35	1.84 5.20 11.32
37	3	41.41 27.38 26.26	13.84 20.51 33.79	1.81 8.48 21.76
38	1	29.23	13.60	1.54
39	2	49.31 33.18	13.60 23.71	1.55 11.67
40	4	36.89 22.42 16.52 18.83	16.48 21.27 26.39 33.23	4.44 9.24 14.36 21.20
41	5	14.03 45.61 30.46 26.76 30.46	0.36 13.60 19.07 28.23 33.99	-11.67 1.56 7.04 16.20 21.96
42	3	30.58 33.01 28.93	13.40 16.95 20.39	0.36 3.92 7.36
43	2	33.70 25.67	13.84 23.51	0.79 10.47
44	3	11.03 32.94 31.16	12.80 15.92 28.07	0.76 3.88 16.03

Suite à la page suivante

TABLE E.9 – *Suite de la page précédente*

N^o stimulation	**Nombre de pics**	**Amplitude des pics (en %)**	**Localisation des pics (en s)**	**Retard par rapport à l'ordre de déglutition (en s)**
45	5	10.67	0.20	-10.83
		37.95	13.96	2.93
		22.67	19.95	8.93
		13.58	25.67	14.64
		25.73	29.91	18.88
46	4	48.59	13.64	0.59
		39.95	17.19	4.15
		15.12	23.99	10.95
		27.03	31.47	18.43
47	3	30.54	15.36	3.32
		17.64	19.39	7.36
		11.52	22.43	10.40
48	3	26.72	0.28	-11.75
		26.86	13.96	1.93
		31.19	30.27	18.24
49	3	48.96	13.72	1.69
		11.26	25.75	13.73
		29.53	30.95	18.92
50	4	43.42	13.60	1.56
		34.45	16.79	4.76
		30.04	19.91	7.88
		28.38	24.23	12.20
51	3	41.98	13.64	1.61
		28.87	24.11	12.09
		31.24	28.67	16.65
52	3	45.62	13.80	3.78
		18.20	20.31	10.30
		31.76	27.87	17.86
53	3	38.34	15.92	3.90
		28.66	19.51	7.50
		31.03	30.23	18.22
54	3	34.29	15.60	4.58
		15.11	21.79	10.78
		26.32	25.75	14.74
55	4	43.65	13.68	3.66
		29.19	18.11	8.10
		30.99	21.95	11.94
		12.93	32.23	22.22
56	1	32.94	16.56	4.54
57	3	42.20	13.68	3.66
		11.91	19.51	9.50
		24.72	32.67	22.66
58	3	38.54	13.84	2.84
		30.18	20.59	9.59
		33.18	34.35	23.35
59	3	38.83	14.04	2.04
		32.55	22.99	11.00
		10.48	34.07	22.08
60	2	37.71	13.52	2.52
		30.46	21.59	10.60

Suite à la page suivante

TABLE E.9 – *Suite de la page précédente*

N^o stimulation	**Nombre de pics**	**Amplitude des pics (en %)**	**Localisation des pics (en s)**	**Retard par rapport à l'ordre de déglutition (en s)**
61	4	46.03	13.32	2.33
		35.61	16.71	5.73
		11.59	25.15	14.17
		27.26	28.83	17.85
62	1	43.48	13.48	1.47
63	4	38.04	13.44	2.43
		23.89	23.15	12.14
		17.57	26.79	15.78
		12.83	30.43	19.42
64	5	36.84	2.64	-10.39
		43.21	13.68	0.65
		20.63	17.99	4.97
		25.10	22.39	9.37
		18.73	31.55	18.53
65	6	10.44	0.52	-11.51
		43.07	13.56	1.53
		13.29	17.07	5.05
		31.62	23.59	11.57
		23.61	27.51	15.49
		21.89	30.75	18.73
66	4	51.05	13.40	2.37
		23.65	18.39	7.37
		10.71	23.75	12.73
		33.81	28.55	17.53
67	4	53.03	14.04	4.02
		40.65	20.43	10.42
		38.76	30.35	20.33
		37.86	33.75	23.73
68	3	47.26	14.00	1.97
		40.06	17.11	5.09
		12.66	32.95	20.92
69	4	10.40	12.84	2.81
		31.83	16.00	5.97
		21.54	22.11	12.09
		21.97	27.99	17.96
70	4	48.65	14.16	2.12
		35.83	20.27	8.24
		13.84	26.15	14.12
		36.28	31.55	19.52
71	4	35.24	13.72	1.69
		16.41	17.59	5.57
		36.85	23.07	11.05
		13.81	26.35	14.33
72	1	46.03	13.68	1.64
73	4	46.59	13.60	0.57
		23.96	18.43	5.41
		16.08	22.75	9.73
		22.74	27.35	14.33
74	4	32.18	13.68	1.65
		32.47	16.87	4.85
		13.91	23.75	11.73
		12.23	30.67	18.65

Suite à la page suivante

TABLE E.9 – *Suite de la page précédente*

N^o stimulation	**Nombre de pics**	**Amplitude des pics (en %)**	**Localisation des pics (en s)**	**Retard par rapport à l'ordre de déglutition (en s)**
75	3	48.35 17.23 24.26	13.48 18.87 25.95	2.45 7.85 14.93
76	4	38.90 28.42 10.37 28.77	13.52 23.11 26.67 31.31	0.49 10.09 13.65 18.29
77	4	26.60 34.49 16.05 31.69	13.96 17.23 24.27 28.35	1.93 5.21 12.25 16.33
78	4	47.29 48.82 36.17 28.75	13.52 16.79 25.83 33.35	1.48 4.76 13.80 21.31
79	3	14.89 45.94 30.01	0.60 13.60 24.15	-10.41 2.58 13.14
80	3	43.77 28.76 14.19	13.16 18.27 32.39	1.16 6.28 20.40

E.2.4 Sujet 4

La figure E.18 montre la moyenne sur toutes les stimulations, la moyenne sur les stimulations avec de la boisson sucrée et la moyenne sur les stimulations avec de l'eau pure. On remarque que contrairement aux trois autres sujets, la moyenne ne montre pas un pic pour l'ouverture de la bouche (flèche bleue). On retrouve cependant, comme pour les autres sujets, un pic vers 13 secondes correspondant à la déglutition (flèche rouge). On trouve peu de différences entre la moyenne sur la boisson sucrée et sur l'eau : le pic de déglutition pour l'eau est légèrement plus élevé et on trouve pour l'eau une augmentation du signal peu après l'ouverture de la bouche alors que le signal diminue pour la boisson sucrée.Le tableau E.10 montre les pics trouvés sur l'ensemble des stimulations. Le pic n^{o}1 se situe pendant la phase d'avertissement et correspond sans doute à l'ouverture de la bouche. Le pic n^{o}2 se situe peu après l'ordre de déglutition : c'est le premier pic de déglutition. Le pic n^{o}3 se situe à la fin de la phase de déglutition et les suivants sont des pics de déglutitions qui se prolongent lors la phase de repos. La table E.13 montre une grande diversité des déglutitions : en effet, les stimulations n^{os} 2, 12, 26, 52 et 56 présentent un pic en début de paradigme, les stimulations n^{os}8, 18, 24, 27, 55 et 59 ont un deuxième pic de déglutition vers la fin de la phase de déglutition ; les stimulations 3, 45, 51, 5, 78 et 80 ont des pics de déglutition pendant la phase de repos. Le reste des stimulations ne présentent que le pic de déglutition.

Ces stimulations se regroupent après un algorithme de parcellisation en 5 parcelles. Le tableau E.12 montre la répartition des stimulations dans les cinq parcelles. La parcelle n^o 1 regroupe des stimulations présentant un pic en début de paradigme. La parcelle n^o 2 ne comporte que la stimulation 10, ce qui montre que cette stimulation est éloignée (au sens de la distance euclidienne) des autres.La figure E.19 montre les centroïdes de chaque parcelle. Les centroïdes des parcelles n^o 3 et 5 ne présentent qu'un seul pic au moment de la déglutition ; un tel lissage s'explique par le grand nombre de stimulations présentes dans ces parcelles. Le centroïde de la parcelle n^o 1 comporte deux pics : le premier un peu après l'ouverture de la bouche et le deuxième correspondant à la déglutition. Le centroïde de la parcelle n^o 2 explique pourquoi la stimulation 10 est

N^o de la parcelle	**N^{os} des stimulations dans la parcelle**
1	21,74
2	78
3	7
4	9,11,49,53
5	12,14,26,43,56,72
6	33,40,59
7	6,18,30,57,62
8	20,27,32
9	22
10	47,69,73
11	35,61
12	75,80
13	23
14	44,66
15	63
16	42,50
17	51,79
18	3,54,60
19	4,8,41
20	45,70,71
21	65
22	13,19,36,48,76
23	46,68,77
24	52,67
25	64
26	2,17,34,37,38,58
27	10,16,24,31
28	29,39
29	15,25,28,55
30	1,5

TABLE E.8 – Répartition des stimulations dans les parcelles pour le sujet 3.

N^o de pic	**Localisation (en s)**	**Amplitude du pic (en %)**
1	1.92±1.10	27.56±12.00
2	13.77±0.26	40.33±3.81
3	17.27±0.72	15.37±5.65
4	20.17±0.20	12.27±3.06
5	27.27±0.80	18.21±5.78
6	32.63±2.04	19.35±12.04

TABLE E.10 – Caractérisation des pics sur l'ensemble des stimulations pour le sujet 4.

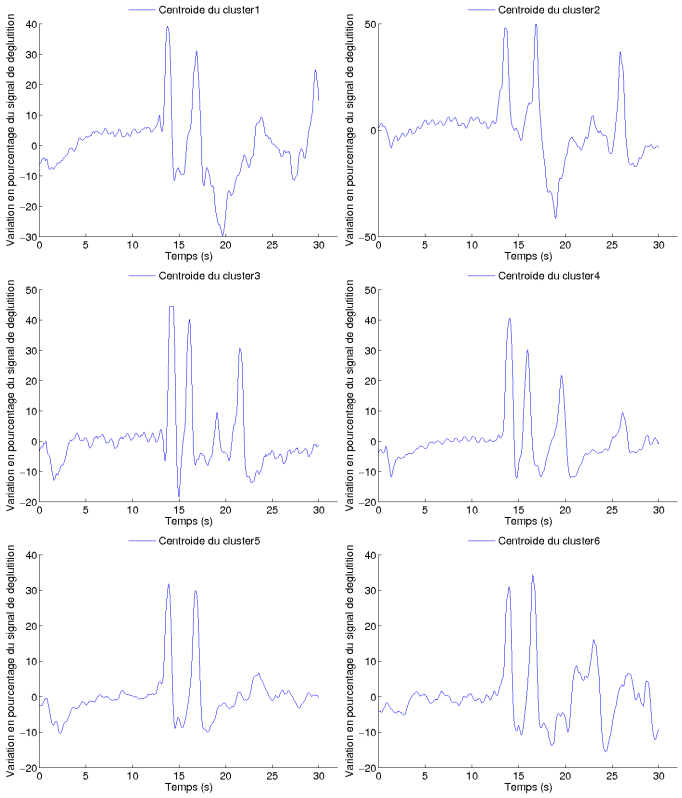


FIGURE E.12 – Centroïdes des parcelles 1 à 6 pour le sujet 3.

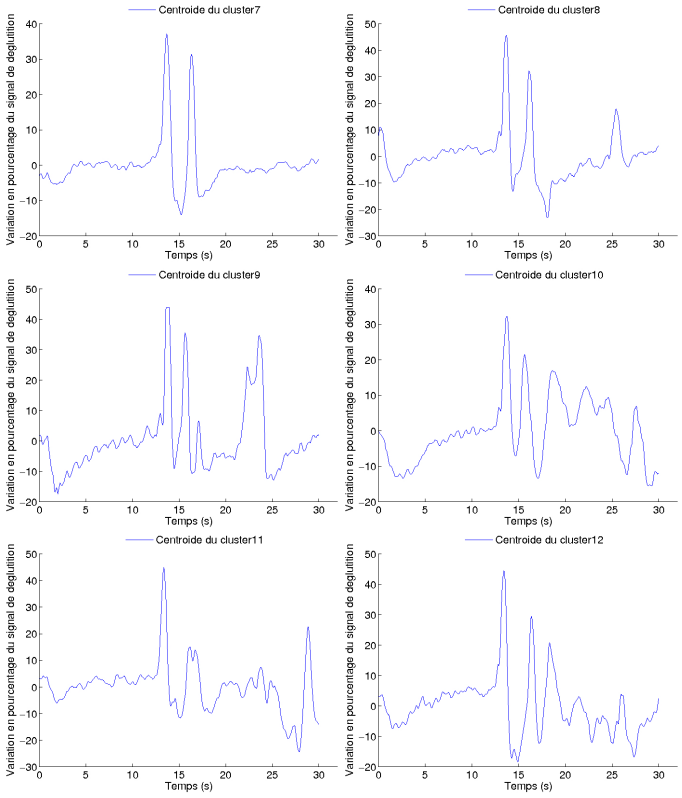


FIGURE E.13 – Centroids des parcelles 7 à 12 pour le sujet 3.

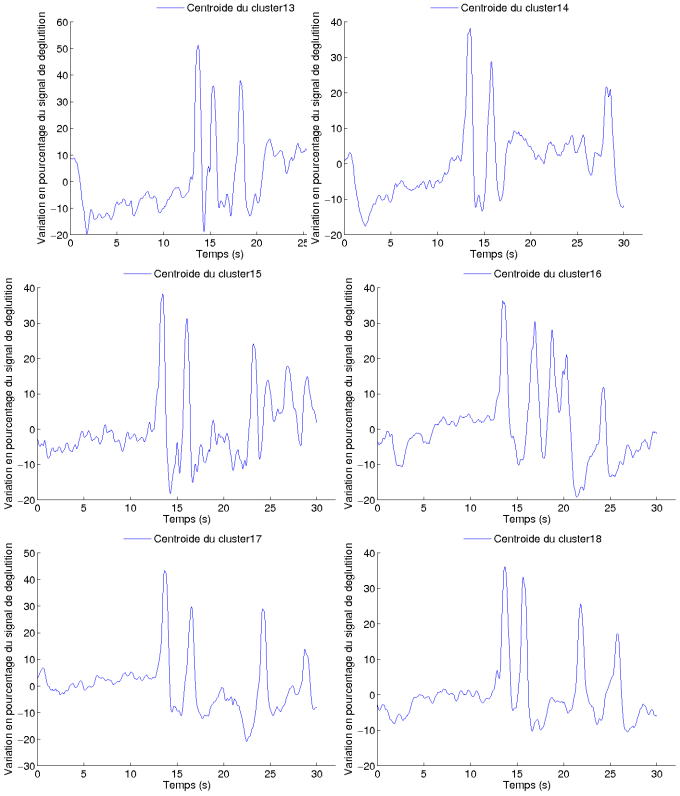


FIGURE E.14 – Centroïdes des parcelles 13 à 18 pour le sujet 3.

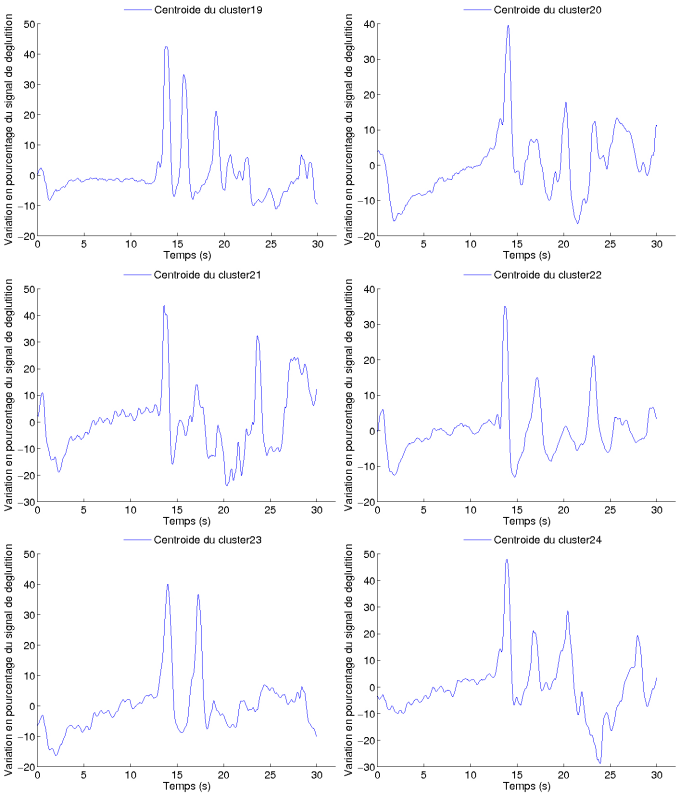


FIGURE E.15 – Centroïdes des parcelles 19 à 24 pour le sujet 3.

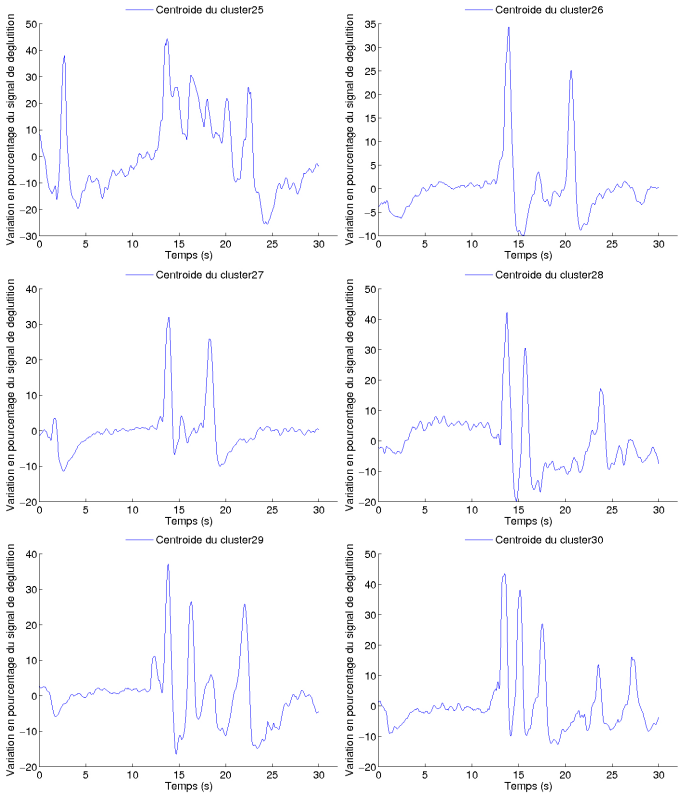
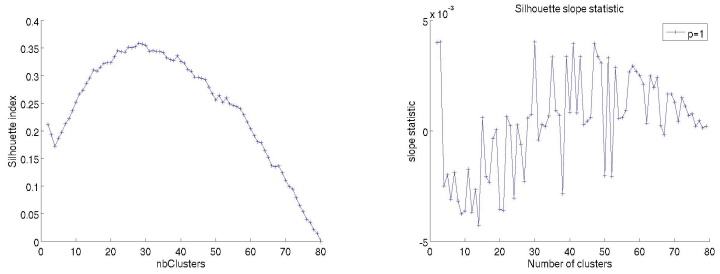


FIGURE E.16 – Centroïdes des parcelles 25 à 30 pour le sujet 3.



(a) Indice de silhouette (b) *Slope statistic*

FIGURE E.17 – Détermination du nombre optimal de parcelles pour le sujet 3.

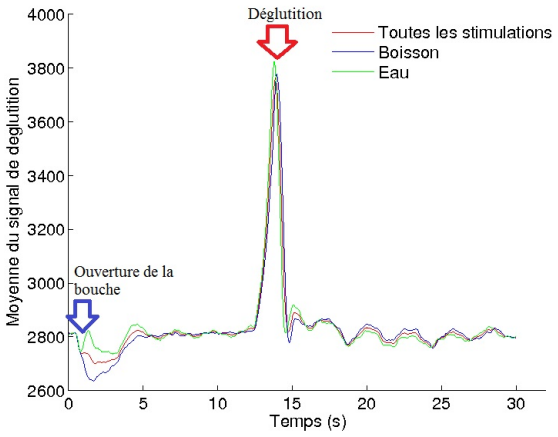


FIGURE E.18 – Moyennes des signaux de déglutition pour toutes les stimulations pour le sujet 4.

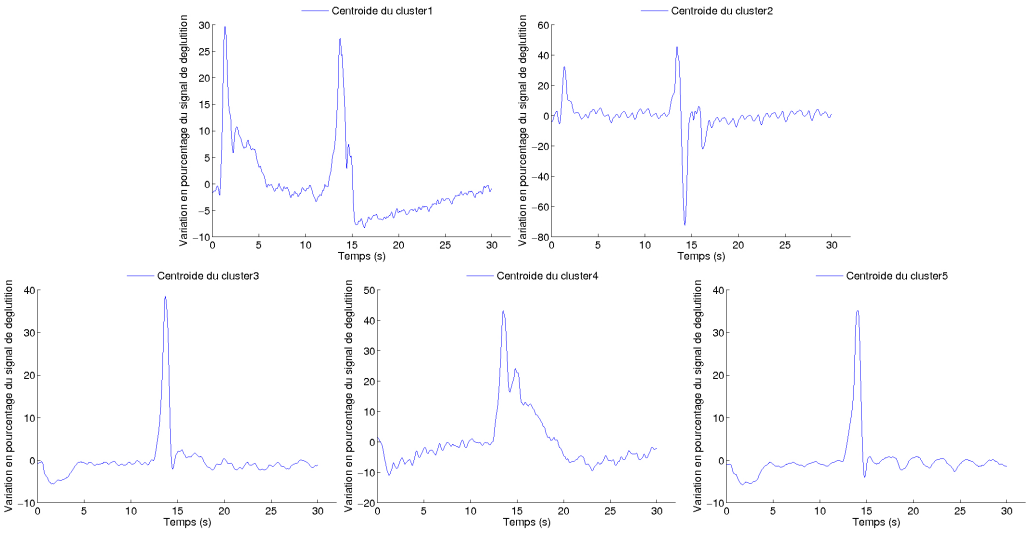


FIGURE E.19 – Centroïdes des parcelles pour le sujet 4.

éloignée (au sens de la distance euclidienne) des autres stimulations : en plus de présenter un pic pour l'ouverture de la bouche, le pic de déglutition est suivi d'une forte diminution de signal. Le centroïde de la parcelle n^{o}4 a deux pics peu après l'ordre de déglutition. Le tableau E.11 montre les pics trouvés sur l'ensemble des centroïdes : on y trouve le pic pour l'ouverture de la bouche et le pic de déglutition.

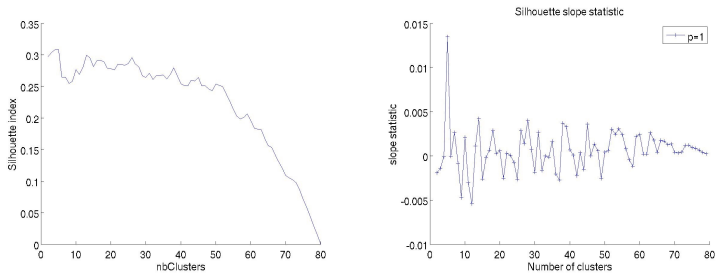
N^o de pic	**Localisation (en s)**	**Amplitude du pic (en %)**
1	1.34±0.03	31.07±1.92
2	13.64±0.25	38.00±7.14

TABLE E.11 – Caractérisation des pics sur l'ensemble des centroïdes pour le sujet 4.

N^o de la parcelle	**N^{os} des stimulations dans la parcelle**
1	2,26,56
2	10
3	1,5,6,9,12,14,16,18,19,22,28,30,32,33,34,36 38,48,54,55,58,60,62,66,68,72,73,74
4	8,13,24,50
5	3,4,7,11,15,17,20,21,23,25,27,29,31,35 37,39,40,41,42,43,44,45,46,47,49,51,52,53 57,59,61,63,64,65,67,69,70,71,75,76,77,78,79,80

TABLE E.12 – Répartition des stimulations dans les parcelles pour le sujet 4.

La figure E.20(a) montre l'indice de silhouette (**cf. équation 6.2**) pour le sujet 4 et la figure E.20(b) illustre la *slope statitic* (**cf. équation 6.7**). La courbe de l'indice de silhouette a une forme de cloche mais lorsqu'on passe à la *slope statistic*, le nombre optimal de parcelles est clairement de cinq avec un score assez élevé. Contrairement aux autres sujets, les autres maxima locaux sont bien en dessous du score pour cinq parcelles.



(a) Indice de silhouette (b) *Slope statistic*

FIGURE E.20 – Détermination du nombre optimal de parcelles pour le sujet 4.

TABLE E.13 – Caractérisation des stimulations selon leur nombre de pics pour le sujet 4.

N^o stimulation	**Nombre de pics**	**Amplitude des pics (en %)**	**Localisation des pics (en s)**	**Retard par rapport à l'ordre de déglutition (en s)**
1	1	43.69	13.40	1.36
2	2	38.83 41.03	1.28 13.76	-10.75 1.73
3	2	43.62 16.19	13.80 26.51	1.76 14.48
4	1	42.82	13.80	1.75
5	1	44.67	13.44	1.40
6	1	42.37	13.48	1.43
7	1	42.26	14.00	1.97
8	2	43.08 20.85	13.48 16.71	1.44 4.68
9	1	33.82	13.64	1.61
10	2	32.23 45.38	1.36 13.40	-10.68 1.35
11	1	44.78	13.76	1.72
12	2	11.46 43.45	3.84 13.20	-8.20 1.15
13	1	49.13	13.32	1.29
14	1	37.78	13.64	1.59
15	1	37.18	14.04	2.00
16	1	39.14	13.68	1.64
17	1	35.21	13.80	1.75
18	2	36.26 10.77	13.68 17.11	1.64 5.08
19	1	41.59	13.60	1.56
20	1	38.16	13.80	1.76
21	1	39.04	13.96	1.91
22	1	31.34	13.72	1.68
23	1	38.20	14.00	1.95
24	2	44.87 13.31	13.44 16.64	1.40 4.60
25	1	38.41	13.96	1.92
26	2	41.54 32.88	1.36 14.56	-10.68 2.51
27	2	38.91 24.04	13.92 18.15	1.88 6.12
28	1	43.71	13.68	1.62

Suite à la page suivante

TABLE E.13 – *Suite de la page précédente*

N^o stimulation	**Nombre de pics**	**Amplitude des pics (en %)**	**Localisation des pics (en s)**	**Retard par rapport à l'ordre de déglutition (en s)**
29	1	42.51	14.24	2.21
30	1	38.74	13.64	1.60
31	1	38.37	14.16	2.11
32	1	37.97	13.60	1.56
33	1	40.99	13.60	1.57
34	1	44.07	13.28	1.23
35	1	39.01	14.16	2.12
36	1	39.26	13.72	1.67
37	1	42.25	14.00	1.96
38	1	39.34	13.48	1.43
39	1	36.62	13.80	1.76
40	1	35.12	13.88	1.84
41	1	43.23	13.72	1.68
42	1	43.94	13.80	2.77
43	1	37.28	14.16	1.11
44	1	36.81	13.76	1.72
45	2	39.60 27.87	13.96 34.07	1.93 22.04
46	1	41.92	13.80	1.76
47	1	41.92	13.96	1.93
48	1	43.64	13.40	1.37
49	1	43.30	13.84	0.80
50	1	44.12	14.76	2.72
51	2	44.15 10.84	13.92 31.19	-0.12 17.15
52	2	23.89 44.80	1.04 13.80	-11.00 1.76
53	1	43.64	13.76	1.73
54	1	33.51	13.64	3.61
55	3	45.04 10.96 10.11	13.56 16.83 20.03	2.54 5.82 9.02
56	2	17.40 43.09	2.64 13.60	-9.38 1.57
57	1	35.17	13.96	2.93
58	1	39.70	13.68	1.65
59	2	34.59 12.30	14.12 18.19	2.09 6.17
60	1	44.52	13.56	0.52
61	1	37.21	13.88	2.84
62	1	38.39	13.68	1.65
63	1	35.52	14.24	4.21
64	1	35.79	13.80	2.77
65	1	35.17	13.80	1.77
66	2	43.42 13.71	13.44 27.19	3.41 17.16
67	1	37.56	13.84	2.81
68	1	37.44	13.44	1.41
69	1	40.76	13.76	0.74
70	1	34.18	13.96	-0.06
71	1	34.42	13.96	1.93

Suite à la page suivante

TABLE E.13 – *Suite de la page précédente*

N^o stimulation	**Nombre de pics**	**Amplitude des pics (en %)**	**Localisation des pics (en s)**	**Retard par rapport à l'ordre de déglutition (en s)**
72	1	45.05	13.56	-0.47
73	1	42.50	13.56	-0.46
74	1	36.69	13.68	1.66
75	1	43.28	13.84	0.82
76	1	44.65	13.84	1.82
77	1	45.49	13.80	1.78
78	2	40.01 14.44	13.84 20.31	1.81 8.29
79	1	43.49	13.80	1.78
80	2	44.09 24.73	13.80 28.11	1.79 16.10

E.3 Matrice de corrélations

FIGURE E.21 – Matrice de corrélations entre les 1000 parcelles pour le sujet n^o 2

Seules les corrélations telles que $|r| \leq 0,3$ sont représentées. Pour aérer les labels des axes, le nom des parcelles avec un numéro pair est donné sur l'axe vertical et le nom des parcelles avec un numéro impair se trouve sur l'axe horizontal. Lorsque son nom n'est pas mentionné, la parcelle est référencée par un numéro.

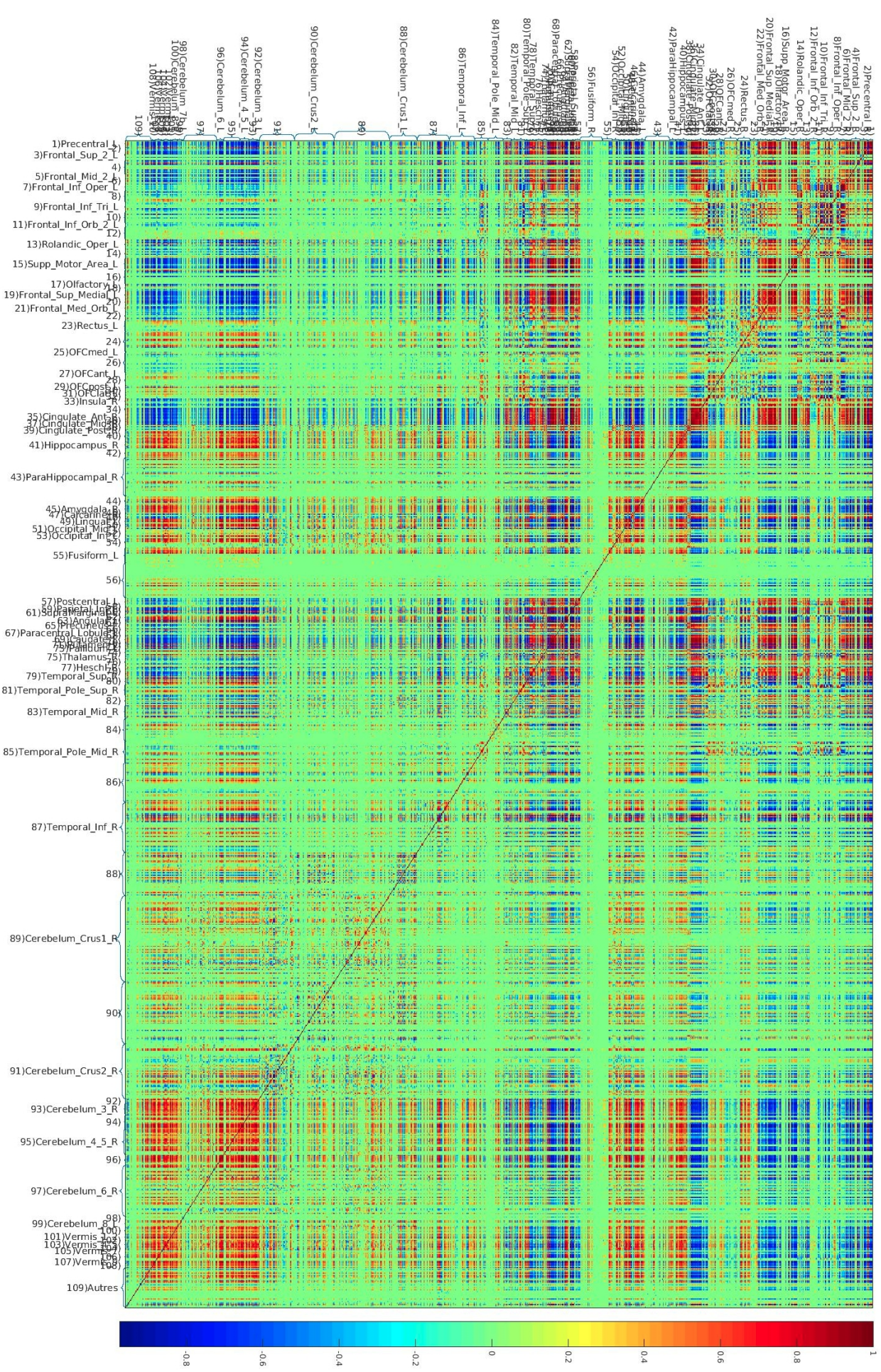


FIGURE E.22 – Matrice de corrélations entre les 1000 parcelles pour le sujet n° 3

Seules les corrélations telles que $|r| \leq 0,3$ sont représentées. Pour aérer les labels des axes, le nom des parcelles avec un numéro pair est donné sur l'axe vertical et le nom des parcelles avec un numéro impair se trouve sur l'axe horizontal. Lorsque son nom n'est pas mentionné, la parcelle est référencée par un numéro.

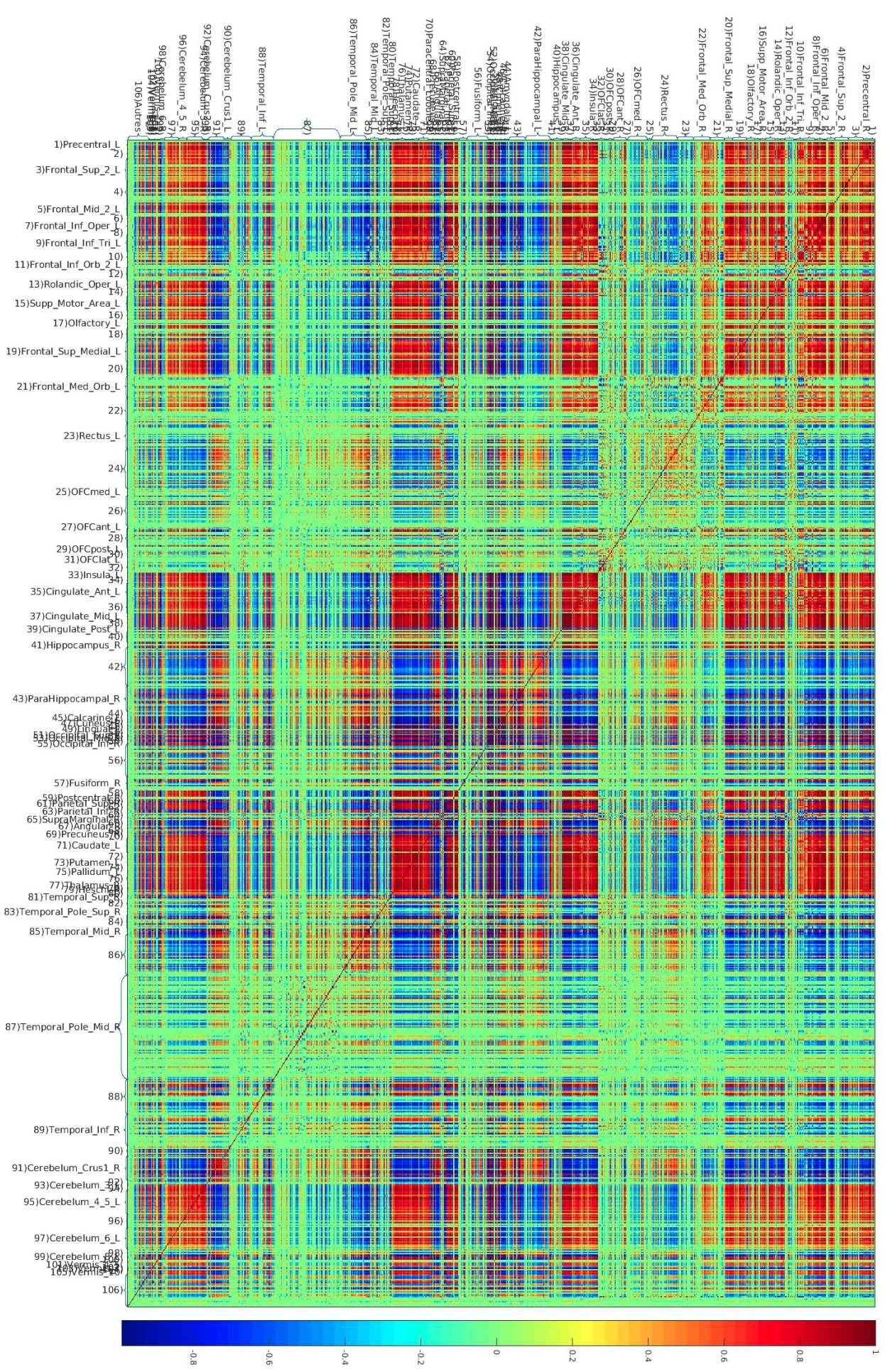


FIGURE E.23 – Matrice de corrélations entre les 1000 parcelles pour le sujet n^o 4

Seules les corrélations telles que $|r| \leq 0,3$ sont représentées. Pour aérer les labels des axes, le nom des parcelles avec un numéro pair est donné sur l'axe vertical et le nom des parcelles avec un numéro impair se trouve sur l'axe horizontal. Lorsque son nom n'est pas mentionné, la parcelle est référencée par un numéro.

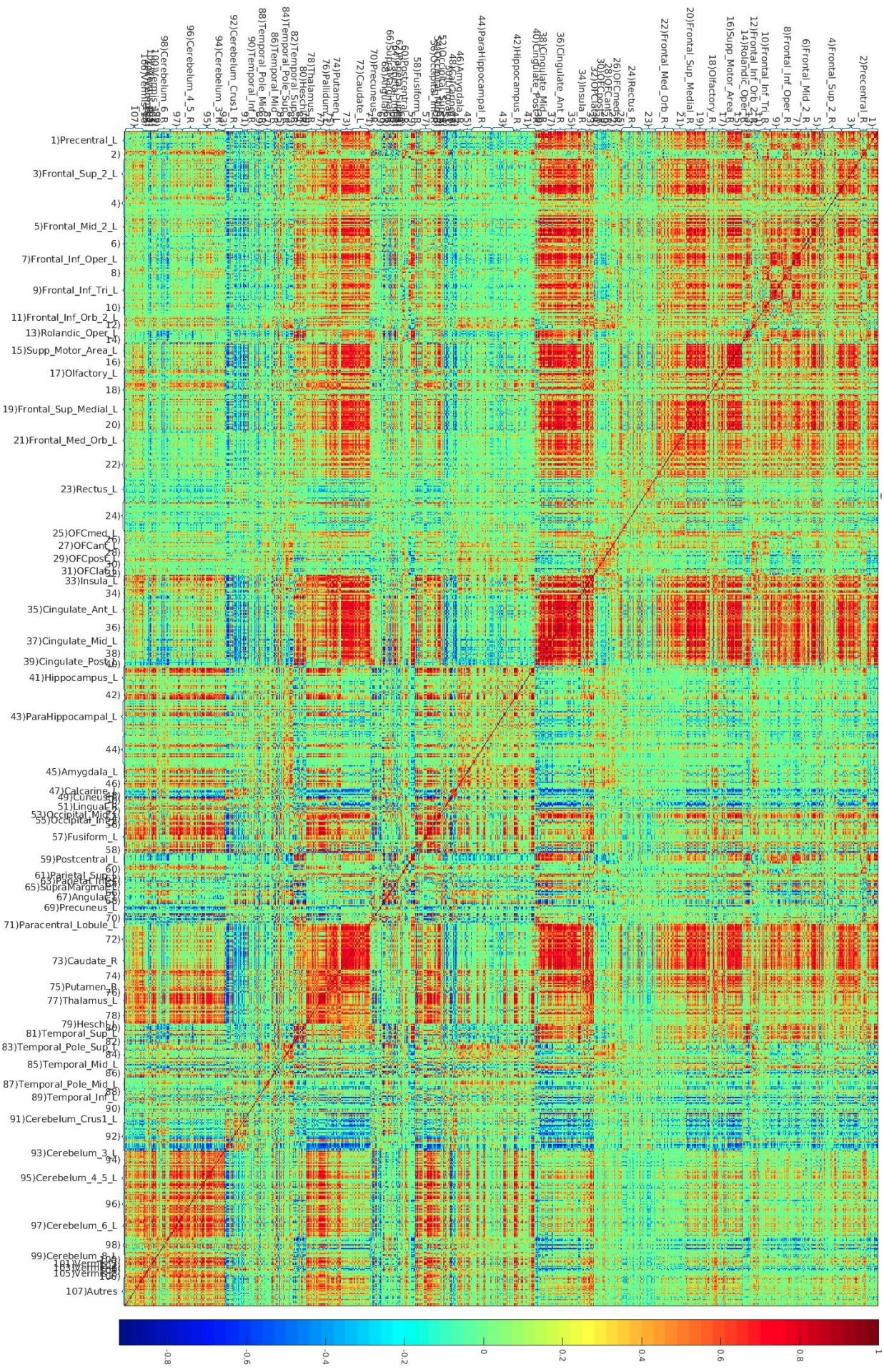


FIGURE E.24 – Matrice de corrélations entre les 1000 parcelles moyennée sur le groupe (parcellisation de groupe)

Seules les corrélations telles que $|r| \leq 0,3$ sont représentées. Pour aérer les labels des axes, le nom des parcelles avec un numéro pair est donné sur l'axe vertical et le nom des parcelles avec un numéro impair se trouve sur l'axe horizontal. Lorsque son nom n'est pas mentionné, la parcelle est référencée par un numéro.

Bibliographie

[1] *Article wikipédia sur l'analyse en composantes indépendantes.* https://fr.wikipedia.org/wiki/Analyse_en_composantes_ind%C3%A9pendantes.

[2] A. Abraham, E. Dohmatob, B. Thirion, D. Samaras, and G. Varoquaux, *Extracting Brain Regions from Rest fMRI with Total-Variation Constrained Dictionary Learning*, in Medical Image Computing and Computer-Assisted Intervention – MICCAI 2013, D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M. Y. Vardi, G. Weikum, K. Mori, I. Sakuma, Y. Sato, C. Barillot, and N. Navab, eds., vol. 8150, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 607–615.

[3] A. Abraham, F. Pedregosa, M. Eickenberg, P. Gervais, A. Mueller, J. Kossaifi, A. Gramfort, B. Thirion, and G. Varoquaux, *Machine learning for neuroimaging with scikit-learn*, Frontiers in Neuroinformatics, 8 (2014).

[4] D. A. Abrams, S. Ryali, T. Chen, P. Chordia, A. Khouzam, D. J. Levitin, and V. Menon, *Inter-subject synchronization of brain responses during natural music listening*, European Journal of Neuroscience, 37 (2013), pp. 1458–1469.

[5] G. Aguirre, E. Zarahn, and M. D'Esposito, *The variability of human, bold hemodynamic responses*, NeuroImage, 8 (1998), pp. 360 – 369.

[6] C. Allefeld and J.-D. Haynes, *Searchlight-based multi-voxel pattern analysis of fMRI by cross-validated MANOVA*, NeuroImage, 89 (2014), pp. 345–357.

[7] M. AltafWani and S. M. K Quadri, *Accelerated Parallel Generation of Binomial Coefficients using GPU*, International Journal of Computer Applications, 71 (2013), pp. 11–13.

[8] P. Amestoy and M. Daydé, *Calcul Réparti et Grid Computing.* http://amestoy.perso.enseeiht.fr/COURS/CRGC.pdf.

[9] F. Andreelli and H. Mosbah, *Irm fonctionnelle cérébrale : les principes*, Médecine des Maladies Métaboliques, 8 (2014), pp. 13 – 19.

[10] M. Angeletti, J.-M. Bonny, F. Durif, and J. Koko, *Parallel hierarchical agglomerative clustering for fmri data*, in Parallel Processing and Applied Mathematics, R. Wyrzykowski, J. Dongarra, E. Deelman, and K. Karczewski, eds., Cham, 2018, Springer International Publishing, pp. 265–275.

[11] G. Ares and P. Varela, eds., *Methods in Consumer Research*, vol. 2, 20188, ch. 3.4.2 Positioning of the Participant and Stimulus Delivery.

[12] S. Arslan, S. I. Ktena, A. Makropoulos, E. C. Robinson, D. Rueckert, and S. Parisot, *Human brain mapping : A systematic comparison of parcellation methods for the human cerebral cortex*, NeuroImage, 170 (2018), pp. 5–30.

[13] S. Arslan and D. Rueckert, *Multi-Level Parcellation of the Cerebral Cortex Using Resting-State fMRI*, in Medical Image Computing and Computer-Assisted Intervention –

MICCAI 2015, N. Navab, J. Hornegger, W. M. Wells, and A. F. Frangi, eds., vol. 9351, Springer International Publishing, Cham, 2015, pp. 47–54.

[14] J. Ashburner, *A fast diffeomorphic image registration algorithm*, NeuroImage, 38 (2007), pp. 95 – 113.

[15] M. Bacha-Trams, E. Glerean, R. Dunbar, J. M. Lahnakoski, E. Ryyppö, M. Sams, and I. P. Jääskeläinen, *Differential inter-subject correlation of brain activity when kinship is a variable in moral dilemma*, Scientific Reports, 7 (2017).

[16] M. Bader and C. Zenger, *Cache oblivious matrix multiplication using an element ordering based on a peano curve*, Linear Algebra and its Applications, 417 (2006), pp. 301–313.

[17] R. Baumgartner, L. Ryner, W. Richter, R. Summers, M. Jarmasz, and R. Somorjai, *Comparison of two exploratory data analysis methods for fMRI : fuzzy clustering vs. principal component analysis*, Magnetic Resonance Imaging, 18 (2000), pp. 89–94.

[18] E. B. Beall and M. J. Lowe, *Isolating physiologic noise sources with independently determined spatial measures*, NeuroImage, 37 (2007), pp. 1286–1300.

[19] C. Beckmann and S. M. Smith, *Tensorial extensions of independent component analysis for multisubject FMRI analysis*, NeuroImage, 2009 (2005).

[20] C. F. Beckmann, *Modelling with independent components*, NeuroImage, (2012).

[21] Y. Behzadi, K. Restom, J. Liau, and T. T. Liu, *A Component Based Noise Correction Method (CompCor) for BOLD and Perfusion Based fMRI*, NeuroImage, 37 (2007), pp. 90–101.

[22] Y. Benjamini and Y. Hochberg, *Controlling the false discovery rate- a pratical and powerful approach for multiple testing*, Journal of the Royal Statistical Society, (1995).

[23] C. M. Bennett, A. A. Baird, M. B. Miller, and G. L. Wolford, *Neural Correlates of Interspecies Perspective Taking in the Post-Mortem Atlantic Salmon : An Argument For Proper Multiple Comparisons Correction*, Journal of Serendipitous and Unexpected Results, (2009).

[24] C. M. Bennett, G. L. Wolford, and M. B. Miller, *The principled control of false positives in neuroimaging*, Tools of the Trade, (2009).

[25] R. M. Birn, P. A. Bandettini, R. W. Cox, A. Jesmanowicz, and R. Shaker, *Magnetic field changes in the human brain due to swallowing or speaking*, Magnetic Resonance in Medicine, 40 (1998), pp. 55–60.

[26] J. M. Bonny, C. Sinding, and T. Thomas-Danguin, *Functional MRI and Sensory Perception of Food*, Springer International Publishing, Cham, 2017, pp. 1–20.

[27] E. N. Brown and M. Behrmann, *Controversy in statistical analysis of functional magnetic resonance imaging data*, Proceedings of the National Academy of Sciences, 114 (2017), pp. E3368–E3369.

[28] A. Buettner, *Observation of the Swallowing Process by Application of Videofluoroscopy and Real-time Magnetic Resonance Imaging–Consequences for Retronasal Aroma Stimulation*, Chemical Senses, 26 (2001), pp. 1211–1219.

[29] E. Bullmore, M. Brammer, S. Rabe-Hesketh, V. Curtis, R. Morris, S. Williams, T. Sharma, and P. McGuire, *Methods for diagnosis and treatment of stimulus-correlated motion in generic brain activation studies using fMRI*, Human Brain Mapping, 7 (1999), pp. 38–48.

[30] A. Buttari, *Multicore and multicore programmaing with openmp.* `http://buttari.perso.enseeiht.fr/docs/sysconc.pdf`.

[31] R. B. Buxton, E. C. Wong, and L. R. Frank, *Dynamics of blood flow and oxygenation changes during brain activation : The balloon model*, Magnetic Resonance in Medicine, 39 (1998), pp. 855–864.

[32] C. Caballero-Gaudes and R. C. Reynolds, *Methods for cleaning the BOLD fMRI signal*, NeuroImage, 154 (2017), pp. 128–149.

[33] V. Calhoun, T. Adali, G. Pearlson, and J. Pekar, *A Method for Making Group Inference from Functional MRI Data Using Indepednent Component Analysis*, Human Brain Mapping, (2001).

[34] V. D. Calhoun, T. Adali, V. B. McGinty, J. Pekar, T. Watson, and G.D.Pearlson, *fMRI Activation in a Visual-Perception Task : Network of Areas Detected Using the General Linear Model and Independent Components Analysis*, NeuroImage, 14 (2001).

[35] V. D. Calhoun, T. Adali, G. D. Pearlson, and J. J. Pekar, *Spatial and temporal independent component analysis of functional MRI data containing a pair of task-related waveforms*, Human brain mapping, 13 (2001), pp. 43–53.

[36] V. D. Calhoun, J. Liu, and T. Adali, *A review of group ICA for fMRI data and ICA for joint inference of imaging, genetic, and ERP data*, NeuroImage, 45 (2009), pp. S163–S172.

[37] J. Carp, *On the plurality of (methodological) worlds : Estimating the analytic flexibility of fmri experiments*, Frontiers in Neuroscience, 6 (2012), p. 149.

[38] ——, *The secret lives of experiments : Methods reporting in the fMRI literature*, NeuroImage, 63 (2012), pp. 289–300.

[39] B. Cerf-Ducastel and C. Murphy, *Validation of a stimulation protocol suited to the investigation of odor-taste interactions with fMRI*, Physiology&Behaviour, 81 (2004).

[40] D. Chang, N. A. Jones, D. Li, and M. Ouyang, *Compute pairwise euclidean distances of data points with GPUs*, (2008).

[41] A. Charpentier, *Article sur la valeur-p.* `http://freakonometrics.hypotheses.org/2462`.

[42] G. Chen, Y.-W. Shin, P. A. Taylor, D. R. Glen, R. C. Reynolds, R. B. Israel, and R. W. Cox, *Untangling the relatedness among correlations, part I : Nonparametric approaches to inter-subject correlation analysis at the group level*, NeuroImage, 142 (2016), pp. 248–259.

[43] G. Chen, P. A. Taylor, and R. W. Cox, *Is the statistic value all we should care about in neuroimaging ?*, NeuroImage, 147 (2017), pp. 952–959.

[44] J. R. Chumbley and K. J. Friston, *False discovery rate revisited : FDR and topolgical inference using Gaussian random field*, NeuroImage, (2009).

[45] J. R. Chumbley, K. Worsley, G. Flandinn, and K. J. Friston, *Topological FDR for neuroimaging*, NeuroImage, (2010).

[46] `https://www.supinfo.com/cours/2ADS/chapitres/02-complexite-algorithmes-recursifs`.

[47] D. Cordes, V. M. Haughton, K. Arfanakis, G. J. Wendt, P. A. Turski, C. H. Moritz, M. A. Quigley, and M. E. Meyerand, *Mapping functionally related regions of brain with functional connectivity mr imaging*, American Journal of Neuroradiology, 21 (2000), pp. 1636–1644.

[48] N. Correa, T. Adali, Y.-O. Li, and V. D. Calhoun, *Comparison of blind source separation algorithms for fmri using a new matlab toolbox : Gift*, in Proceedings. (ICASSP '05). IEEE International Conference on Acoustics, Speech, and Signal Processing, 2005., vol. 5, mar 2005, pp. v/401–v/404 Vol. 5.

[49] R. W. Cox, G. Chen, D. R. Glen, R. C. Reynolds, and P. A. Taylor, *fMRI clustering and false-positive rates*, Proceedings of the National Academy of Sciences, 114 (2017), pp. E3370–E3371.

[50] R. C. Craddock, G. James, P. E. Holtzheimer, X. P. Hu, and H. S. Mayberg, *A whole brain fMRI atlas generated via spatially constrained spectral clustering*, Human Brain Mapping, 33 (2012), pp. 1914–1928.

[51] *NVIDIA CUDA C Programming Guide.* `https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html`, 2012.

[52] P. Czarnul, *Parallelization of large vector similarity computations in a hybrid CPU+GPU environment*, The Journal of Supercomputing, 74 (2018), pp. 768–786.

[53] J. R. Dalenberg, H. R. Hoogeveen, R. J. Renken, D. R. Langers, and G. J. ter Horst, *Functional specialization of the male insula during taste perception*, NeuroImage, 119 (2015), pp. 210–220.

[54] M. Dash, S. Petrutiu, and P. Scheuermann, *Efficient parallel hierarchical clustering*, in European Conference on Parallel Processing, Springer, 2004, pp. 363–371.

[55] I. Daubechies, E. Roussos, S. Takerkart, M. Benharrosh, C. Golden, K. D'ardenne, W. Richter, J. D. Cohen, and J. Haxby, *Independent component analysis for brain fMRI does not select for independence*, Proceedings of the National Academy of Sciences, 106 (2009), pp. 10415–10422.

[56] E. Dayan and L. G. Cohen, *Neuroplasticity Subserving Motor Skill Learning*, Neuron, 72 (2011), pp. 443–454.

[57] B. Deen and K. Pelphrey, *Perspective : Brain scans need a rethink*, Nature, 491 (2012).

[58] S. Dehaen, *Neuroimagerie cognitive : principes et limites.* `https://www.college-de-france.fr/media/stanislas-dehaene/UPL1267245260781049365_Cours_1.pdf`.

[59] Z. Du and F. Lin, *A novel parallelization approach for hierarchical clustering*, Parallel Computing, 31 (2005), pp. 523–527.

[60] V. Eijkhout, *Introduction to High Performance Scientific Computing.* `http://pages.tacc.utexas.edu/~eijkhout/Articles/EijkhoutIntroToHPC.pdf`.

[61] A. Eklund, M. Andersson, C. Josephson, M. Johannesson, and H. Knutsson, *Does parametric fMRI analysis with SPM yield valid results ?—An empirical study of 1484 rest datasets*, NeuroImage, 61 (2012), pp. 565–578.

[62] A. Eklund, T. E. Nichols, and H. Knutsson, *Cluster failure : Why fMRI inferences for spatial extent have inflated false-positive rates*, Proceedings of the National Academy of Sciences, 113 (2016), pp. 7900–7905.

[63] ——, *Reply to Brown and Behrmann, Cox, et al., and Kessler et al. : Data and code sharing is the way forward for fMRI*, Proceedings of the National Academy of Sciences, 114 (2017), pp. E3374–E3375.

[64] `http://www.nbertagnolli.com/jekyll/update/2015/12/10/Elbow.html`.

[65] E. B. Erhardt, S. Rachakonda, E. J. Bedrick, E. A. Allen, T. Adali, and V. D. Calhoun, *Comparison of Multi-Subject ICA Methods for Analysis of fMRI Data ?*, Human Brain Mapping, (2011).

[66] F. Esposito, T. Scarabino, A. Hyvarinen, J. Himberg, E. Formisano, S. Comani, G. Tedeschi, R. Goebel, E. Seifritz, and F. D. Salle, *Independent component analysis of fMRI group studies by self-organizing clustering*, NeuroImage, (2005).

[67] J. Etancelin. REIMS GPU SPRING SCHOOL 2016, 2016.

[68] D. A. Feinberg, S. Moeller, S. M. Smith, E. Auerbach, S. Ramanna, M. F. Glasser, K. L. Miller, K. Ugurbil, and E. Yacoub, *Multiplexed echo planar imaging for sub-second whole brain fmri and fast diffusion imaging*, PLOS ONE, 5 (2010), pp. 1–11.

[69] G. Flandin, F. Kherif, X. Pennec, G. Malandain, N. Ayache, and J.-B. Poline, *Improved Detection Sensitivity in Functional MRI Data Using a Brain Parcelling Technique*, in Medical Image Computing and Computer-Assisted Intervention — MICCAI 2002, G. Goos, J. Hartmanis, J. van Leeuwen, T. Dohi, and R. Kikinis, eds., vol. 2488, Springer Berlin Heidelberg, Berlin, Heidelberg, 2002, pp. 467–474.

[70] M. Frigo, C. E. Leiserson, H. Prokop, and S. Ramachandran, *Cache-oblivious algorithms*, in 40th Annual Symposium on Foundations of Computer Science, FOCS '99, 17-18 October, 1999, New York, NY, USA, 1999, pp. 285–298.

[71] K. J. Friston, S. Williams, R. Howard, R. S. J. Frackowiak, and R. Turner, *Movement-Related effects in fMRI time-series*, Magnetic Resonance in Medicine, 35 (1996), pp. 346–355.

[72] A. Fujita, D. Y. Takahashi, A. G. Patriota, and J. R. Sato, *A non-parametric statistical test to compare clusters with applications in functional magnetic resonance imaging data*, Statistics in Medicine, 33 (2014), pp. 4949–4962.

[73] G. Gaggioni, P. Maquet, C. Schmidt, D.-J. Dijk, and G. Vandewalle, *Neuroimaging, cognition, light and circadian rhythms*, Frontiers in Systems Neuroscience, 8 (2014).

[74] J.-H. Gao and S.-H. Yee, *Iterative temporal clustering analysis for the detection of multiple response peaks in fMRI*, Magnetic Resonance Imaging, 21 (2003), pp. 51–53.

[75] X. Gao, T. Zhang, and J. Xiong, *Comparison between spatial and temporal independent component analysis for blind source separation in fMRI data*, in Biomedical Engineering and Informatics (BMEI), 2011 4th International Conference on, vol. 2, IEEE, 2011, pp. 690–692.

[76] G. H. Glover, *Overview of functional magnetic resonance imaging*, Neurosurgery clinics of North America, (2011).

[77] G. H. Glover, T.-Q. Li, and D. Ress, *Image-based method for retrospective correction of physiological motion effects in fMRI : RETROICOR*, Magnetic Resonance in Medicine, 44 (2000), pp. 162–167.

[78] Y. Golland, S. Bentin, H. Gelbard, Y. Benjamini, R. Heller, Y. Nir, U. Hasson, and R. Malach, *Extrinsic and Intrinsic Systems in the Posterior Cortex of the Human Brain Revealed during Natural Sensory Stimulation*, Cerebral Cortex, 17 (2007).

[79] Y. Golland, P. Golland, S. Bentin, and R. Malach, *Data-driven clustering reveals a fundamental subdivision of the human cortex into two global system*, Neuropsychologia, 46 (2008).

[80] J. Gonzalez-Castillo, G. Chen, T. E. Nichols, and P. A. Bandettini, *Variance decomposition for single-subject task-based fMRI activity estimates across many sessions*, NeuroImage, 154 (2017), pp. 206–218.

[81] J. Gonzalez-Castillo, Z. S. Saad, D. A. Handwerker, S. J. Inati, N. Brenowitz, and P. A. Bandettini, *Whole-brain, time-locked activation with simple tasks revealed using massive averaging and model-free analysis*, Proceedings of the National Academy of Sciences, 109 (2012), pp. 5487–5492.

[82] S. N. Goodman, D. Fanelli, and J. P. A. Ioannidis, *What does research reproducibility mean ?*, Science Translational Medicine, 8 (2016), pp. 341ps12–341ps12.

[83] T. K. Goto, A. W. K. Yeung, J. L. K. Suen, B. S. K. Fong, and Y. Ninomiya, *High resolution time–intensity recording with synchronized solution delivery system for the human dynamic taste perception*, Journal of Neuroscience Methods, 245 (2015), pp. 147–155.

[84] L. Gouarin, V. Louvet, and L. Series, *Formation en Calcul Scientifique - LIEM2i - Introduction au calcul parallèle.* `http://calcul.math.cnrs.fr/Documents/Ecoles/LEM2I/Mod3/paral.pdf`.

[85] T. Grootswagers, S. G. Wardle, and T. A. Carlson, *Decoding Dynamic Brain Patterns from Evoked Responses : A Tutorial on Multivariate Pattern Analysis Applied to Time Series Neuroimaging Data*, Journal of Cognitive Neuroscience, 29 (2017), pp. 677–697.

[86] Q. Guo, M. Parlar, W. Truong, G. Hall, L. Thabane, M. McKinnon, R. Goeree, and E. Pullenayegum, *The Reporting of Observational Clinical Functional Magnetic Resonance Imaging Studies : A Systematic Review*, PLoS ONE, 9 (2014), p. e94412.

[87] J. V. Hajnal, R. Myers, A. Oatridge, J. E. Schwieso, I. R. Young, and G. M. Bydder, *Artifacts due to stimulus correlated motion in functional imaging of the brain*, Magnetic Resonance in Medicine, 31 (1994), pp. 283–291.

[88] K. Hamandi, A. Salek Haddadi, A. Liston, H. Laufs, D. Fish, and L. Lemieux, *fMRI temporal clustering analysis in patients with frequent interictal epileptiform discharges : Comparison with EEG-driven analysis*, NeuroImage, 26 (2005), pp. 309–316.

[89] D. A. Handwerker, J. M. Ollinger, and M. D'Esposito, *Variation of bold hemodynamic responses across subjects and brain regions and their effects on statistical analyses*, NeuroImage, 21 (2004), pp. 1639 – 1651.

[90] S. J. Hanson and B. M. Bly, *The distribution of BOLD susceptibility effects in the brain is non-Gaussian :*, Neuroreport, 12 (2001), pp. 1971–1977.

[91] U. Hasson, G. Avidan, H. Gelbard, I. Vallines, M. Harel, N. Minshew, and M. Behrmann, *Shared and idiosyncratic cortical activation patterns in autism revealed under continuous real-life viewing conditions*, Autism Research, 2 (2009), pp. 220–231.

[92] U. Hasson, Y. Nir, I. Levy, G. Fuhrmann, and R. Malach, *Intersubject Synchronization of Cortical Activity During Natural Vision*, ScienceMag, 303 (2004).

[93] U. Hasson, E. Yang, I. Vallines, D. J. Heeger, and N. Rubin, *A Hierarchy of Temporal Receptive Windows in Human Cortex*, The Journal of Neuroscience, 28 (2008).

[94] J.-D. Haynes, *A Primer on Pattern-Based Approaches to fMRI : Principles, Pitfalls, and Perspectives*, Neuron, 87 (2015), pp. 257–270.

[95] M. N. Hebart and C. I. Baker, *Deconstructing multivariate decoding for the study of brain function*, NeuroImage, (2017).

[96] M. N. Hebart, K. Görgen, and J.-D. Haynes, *The Decoding Toolbox (TDT) : a versatile software package for multivariate analyses of functional imaging data*, Frontiers in Neuroinformatics, 8 (2015).

[97] R. Heller, D. Stanley, D. Yekutieli, N. Rubin, and Y. Benjamini, *Cluster-based analysis of fMRI data*, NeuroImage, (2006).

[98] A. Herbec, J.-P. Kauppi, C. Jola, J. Tohka, and F. E. Pollick, *Differences in fMRI intersubject correlation while viewing unedited and edited videos of dance performance*, Cortex, 71 (2015), pp. 341–348.

[99] M. Hodgson, R. S. T. Linforth, and A. J. Taylor, *Simultaneous Real-Time Measurements of Mastication, Swallowing, Nasal Airflown and Aroma Release*, Journal of Agricultural and Food Chemistry, 51 (2003).

[100] A. Hyvarinen, *Independent component analysis : recent advances*, Philosophical Transactions of the Royal Society A : Mathematical, Physical and Engineering Sciences, 371 (2012), pp. 20110534–20110534.

[101] A. Hyvärinen and E. Oja, *Independent component analysis : algorithms and applications*, Neural networks, 13 (2000), pp. 411–430.

[102] A. Hyvärinen and P. Ramkumar, *Testing independent component patterns by inter-subject or inter-session consistency*, Frontiers in Human Neuroscience, (2013).

[103] `http://fmri.ucsd.edu/Howto/3T/structure.html`.

[104] *Reims gpu spring school 2016 :intoduction*. REIMS GPU SPRING SCHOOL 2016, 2016.

[105] J. P. A. Ioannidis, *Why Most Published Research Findings Are False*, PLoS Medicine, 2 (2005), p. 6.

[106] J. Iranpour, G. Morrot, B. Claise, B. Jean, and J.-M. Bonny, *Using High Spatial Resolution to Improve BOLD fMRI Detection at 3t*, PLOS ONE, 10 (2015), pp. 1–15.

[107] G. A. James, O. Hazaroglu, and K. A. Bush, *A human brain atlas derived via n-cut parcellation of resting-state and task-based fMRI data*, Magnetic Resonance Imaging, 34 (2016), pp. 209–218.

[108] P. Jezzard and R. S. Balaban, *Correction for geometric distortion in echo planar images from b0 field variations*, Magnetic Resonance in Medicine, 34, pp. 65–73.

[109] Johnson and Wichern, *Applied Multivariate Statistical Anlaysis*, 2007, ch. 4.7 Detecting Outliers and Cleanng Data.

[110] I. P. Jääskeläinen, K. Koskentalo, M. H. Balk, T. Autti, J. Kauramäki, C. Pomren, and M. Sams, *Inter-Subject Synchronization of Prefontal Cortex Hemodynamic Activity During Natural Viewing*, The Opening Neuroimaging Journal, (2008).

[111] T. Kahnt, L. J. Chang, S. Q. Park, J. Heinzle, and J.-D. Haynes, *Connectivity-Based Parcellation of the Human Orbitofrontal Cortex*, Journal of Neuroscience, 32 (2012), pp. 6240–6250.

[112] S. Kalus, L. Bothmann, C. Yassouridis, M. Czisch, P. G. Sämann, and L. Fahrmeir, *Statistical modeling of time-dependent fMRI activation effects : Time-Dependent fMRI Activation Effects*, Human Brain Mapping, 36 (2015), pp. 731–743.

[113] Y. N. Kami, T. K. Goto, K. Tokumori, T. Yoshiura, K. Kobayashi, Y. Nakamura, H. Honda, Y. Ninomiya, and K. Yoshiura, *The development of a novel automated taste stimulus delivery system for fmri studies on the human cortical segregation of taste*, Journal of Neuroscience Methods, 172 (2008), pp. 48 – 53.

[114] Kauppi, *Inter-subject correlation of brain hemodynamic responses during watching a movie : localization in space and frequency*, Frontiers in Neuroinformatics, (2010).

[115] J.-P. Kauppi, J. Pajula, and J. Tohka, *A versatile software package for inter-subject correlation based analyses of fMRI*, Frontiers in Neuroinformatics, 8 (2014).

[116] D. Kessler, M. Angstadt, and C. S. Sripada, *Reevaluating "cluster failure" in fMRI using nonparametric control of the false discovery rate*, Proceedings of the National Academy of Sciences, 114 (2017), pp. E3372–E3373.

[117] S. Kim and M. Ouyang, *Compute Distance Matrices with GPU*, Global Science Technology Forum, Sept. 2012.

[118] K. K. Kwong, J. W. Belliveau, D. A. Chesler, I. E. Goldberg, R. M. Weisskoff, B. P. Poncelet, D. N. Kennedy, B. E. Hoppel, M. S. Cohen, and R. Turner, *Dynamic magnetic resonance imaging of human brain activity during primary sensory stimulation.*, Proceedings of the National Academy of Sciences, 89 (1992), pp. 5675–5679.

[119] G. N. Lance and W. T. Williams, *A general theory of classificatory sorting strategies 1. hierarchical systems*, The Computer Journal, 9 (1967), pp. 373–380.

[120] D. Langlois, S. Chartier, and D. Gosselin, *An introduction to independent component analysis : InfoMax and FastICA algorithms*, Tutorials in Quantitative Methods for Psychology, 6 (2010), pp. 31–38.

[121] B. Lenoski, L. C. Baxter, L. J. Karam, J. Maisog, and J. Debbins, *On the Performance of Autocorrelation Estimation Algorithms for fMRI Analysis*, IEEE Journal of Selected Topics in Signal Processing, 2 (2008), pp. 828–838.

[122] N. Leonardi, J. Richiardi, M. Gschwind, S. Simioni, J.-M. Annoni, M. Schluep, P. Vuilleumier, and D. Van De Ville, *Principal components of functional connectivity : A new approach to study dynamic brain connectivity during rest*, NeuroImage, 83 (2013), pp. 937–950.

[123] Y. Lerner, C. J. Honey, L. J. Silbert, and U. Hasson, *Topographic Mapping of a Hierarchy of Temporal Receptive Windows Using a Narrated Story*, The Journal of Neuroscience, (2011).

[124] Q. Li, V. Kecman, and R. Salman, *A Chunking Method for Euclidean Distance Matrix Calculation on Large Dataset Using Multi-GPU*, IEEE, Dec. 2010, pp. 208–213.

[125] X. Li and Z. Fang, *Parallel clustering algorithms*, Parallel Computing, 11 (1989), pp. 275–290.

[126] A. Lillywhite, D. Glowinski, A. Camurri, and F. E. Pollick, *Using fMRI and intersubject correlation to explore brain activity during audiovisual observation of a string quartet*, Frontiers in Human Neuroscience, (2015).

[127] M. A. Lindquist, *The Statistical Analysis of fMRI Data*, Statistical Science, (2008).

[128] M. A. Lindquist, J. Meng Loh, L. Y. Atlas, and T. D. Wager, *Modeling the hemodynamic response function in fMRI : Efficiency, bias and mis-modeling*, NeuroImage, 45 (2009), pp. S187–S198.

[129] T. T. Liu, *Noise contributions to the fMRI signal : An overview*, NeuroImage, 143 (2016), pp. 141–151.

[130] T. T. Liu, Y. Behzadi, K. Restom, K. Uludag, K. Lu, G. T. Buracas, D. J. Dubowitz, and R. B. Buxton, *Caffeine alters the temporal dynamics of the visual BOLD response*, NeuroImage, 23 (2004), pp. 1402–1413.

[131] Y. Liu, J.-H. Gao, H.-L. Liu, and P. T. Fox, *The temporal response of the brain after eating revealed by functional MRI*, Nature, 405 (2000), pp. 1055–1058.

[132] N. K. Logothethis, *What we can do and what we cannot do with fmri*, 453 (2008), pp. 869–878.

[133] N. K. Logothetis, *The Underpinnings of the BOLD Functional Magnetic Resonance Imaging Signal*, The Journal of Neuroscience, 23 (2003), pp. 3963–3971.

[134] J. M. Loh, M. Lindquist, and T. Wager, *Residual analysis for detecting mis-modeling in fmri*, 18 (2008), pp. 1421–1448.

[135] J. Loula, G. Varoquaux, and B. Thirion, *Decoding fmri activity in the time domain improves classification performance*, NeuroImage, 180 (2018), pp. 203 – 210. New advances in encoding and decoding of brain signals.

[136] N. Lu, B.-C. Shan, J.-Y. Xu, W. Wang, and K.-C. Li, *An improved temporal clustering analysis method applied to whole-brain data in fMRI study*, Magnetic Resonance Imaging, 25 (2007), pp. 57–62.

[137] W.-L. Luo and T. E. Nichols, *Diagnosis and exploration of massively univariate neuroimaging models*, NeuroImage, 19 (2003), pp. 1014–1032.

[138] D. J. Madden, M. C. Costello, N. A. Dennis, S. W. Davis, A. M. Shepler, J. Spaniol, B. Bucur, and R. Cabeza, *Adult age differences in functional connectivity during executive control*, NeuroImage, 52 (2010), pp. 643 – 657.

[139] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. `http://nlp.stanford.edu/IR-book/html/htmledition/single-link-and-complete-link-clustering-1.html`, 2008.

[140] J. Marchini and A. Presanis, *Comparing methods of analyzing fMRI statistical parametric maps*, NeuroImage, (2004).

[141] F. D. Martino, F. Gentile, F. Esposito, M. Balsi, F. D. Salle, R. Goebel, and E. Formisano, *Classification of fMRI independent components using IC-fingerprints and support vector machine classifiers*, NeuroImage, (2007).

[142] J. F. Matias Rodrigues and C. von Mering, *HPC-CLUST : distributed hierarchical clustering for large sets of nucleotide sequences*, Bioinformatics, 30 (2014), pp. 287–288.

[143] M. J. McKeown, L. K. Hansen, and T. J. Sejnowski, *Independent component analysis of functional mri : what is signal and what is noise ?*, Current opinion in Neurobiology, (2003).

[144] M. J. McKeown, S. Makeig, G. G. Brown, T.-P. Jung, S. S. Kindermann, A. J. Bell, and T. J. Sejnowski, *Analysis of fMRI Data by Blind Separation Into Independant Spatial Components*, Human Brain Mapping, (1998).

[145] M. J. McKeown, T. J. Sejnowski, and others, *Independent component analysis of fMRI data : examining the assumptions*, Human brain mapping, 6 (1998), pp. 368–372.

[146] V. Michel, A. Gramfort, G. Varoquaux, E. Eger, C. Keribin, and B. Thirion, *A supervised clustering approach for fMRI-based inference of brain states*, Pattern Recognition, 45 (2012), pp. 2041–2049.

[147] M. Monti, *Statistical analysis of fmri time-series : A critical review of the glm approach*, Frontiers in Human Neuroscience, 5 (2011), p. 28.

[148] V. L. Morgan, R. R. Price, A. Arain, P. Modur, and B. Abou-Khalil, *Resting functional MRI with temporal clustering analysis for localization of epileptic activity without EEG*, NeuroImage, 21 (2004), pp. 473–481.

[149] C. H. Moritz, B. P. Rogers, and M. E. Meyerand, *Power Spectrum Ranked Independent Component Analysis of a Periodic fMRI Complex Motor Paradigm*, Human Brain Mapping, (2003).

[150] G. Morrot, J.-M. Bonny, B. Lehallier, and M. Zanca, *fMRI of human olfaction at the individual level : Interindividual variability*, Journal of Magnetic Resonance Imaging, 37 (2013), pp. 92–100.

[151] *MPI-une bibliothèque de communication par messages.* https://www-master.ufr-info-p6.jussieu.fr/2007/Ajouts/Master_esj20_2007_2008/IMG/pdf/MPI_pres.pdf.

[152] J. A. Mumford, J.-B. Poline, and R. A. Poldrack, *Orthogonalization of Regressors in fMRI Models*, PLOS ONE, 10 (2015), p. e0126255.

[153] Y. Nakamura, T. K. Goto, K. Tokumori, T. Yoshiura, K. Kobayashi, Y. Nakamura, H. Honda, Y. Ninomiya, and K. Yoshiura, *The temporal change in the cortical activations due to salty and sweet tastes in humans : fmri and time-intensity sensory evaluation.*, Neuroreport, 23 6 (2012), pp. 400–4.

[154] D. E. Nee, *2nd Level (Group) General linear model*, (2014), p. 32.

[155] T. Nichols and S. Hayasaka, *Controlling the familywise error rate in functional neuroimaging : a comparative review*, Statistical Methods in Medical Research, (2003).

[156] W. S. Noble, *How does multiple testing correction work ?*, NATURE BIOTECHNOLOGY, (2009).

[157] L. Nummenmaa, E. Glerean, M. Viinikainen, I. P. Jaaskelainen, R. Hari, and M. Sams, *Emotions promote social interaction by synchronizing brain activity across individuals*, Proceedings of the National Academy of Sciences, 109 (2012), pp. 9599–9604.

[158] S. Ogawa, T. M. Lee, A. R. Kay, and D. W. Tank, *Brain magnetic resonance imaging with contrast dependent on blood oxygenation*, Proceedings of the National Academy of Sciences, 87 (1990), pp. 9868–9872.

[159] C. F. Olson, *Parallel algorithms for hierarchical clustering*, Parallel computing, 21 (1995), pp. 1313–1325.

[160] P. Orban, J. Doyon, M. Petrides, M. Mennes, R. Hoge, and P. Bellec, *The Richness of Task-Evoked Hemodynamic Responses Defines a Pseudohierarchy of Functionally Meaningful Brain Networks*, Cerebral Cortex, 25 (2015), pp. 2658–2669.

[161] J. Pajula, J.-P. Kauppi, and J. Tohka, *Inter-Subject Correlation in fMRI : Method Validation against Stimulus-Model Based Analysis*, PLoS ONE, (2012).

[162] J. Pajula and J. Tohka, *Effects of spatial smoothing on inter-subject correlation based analysis of FMRI*, Magnetic Resonance Imaging, 32 (2014), pp. 1114–1124.

[163] ——, *How Many Is Enough ? Effect of Sample Size in Inter-Subject Correlation Analysis of fMRI*, Computational Intelligence and Neuroscience, 2016 (2016), pp. 1–10.

[164] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, *Scikit-learn : Machine learning in Python*, Journal of Machine Learning Research, 12 (2011), pp. 2825–2830.

[165] V. Perlbarg, P. Bellec, J.-L. Anton, M. Pélégrini-Issac, J. Doyon, and H. Benali, *CORSICA : correction of structured noise in fMRI by automatic identification of ICA components*, Magnetic Resonance Imaging, 25 (2007), pp. 35–46.

[166] J. C. Pfeiffer, T. A.Hollowood, J. Hort, and A. J.taylor, *Temporal Synchrony and Integration of Sub-threshold Taste and Smell Signals*, Chem Senses, (2005).

[167] R. A. Poldrack, C. I. Baker, J. Durnez, K. J. Gorgolewski, P. M. Matthews, M. R. Munafò, T. E. Nichols, J.-B. Poline, E. Vul, and T. Yarkoni, *Scanning the horizon : towards transparent and reproducible neuroimaging research*, Nature Reviews Neuroscience, 18 (2017), pp. 115–126.

[168] J.-B. Poline and M. Brett, *The general linear model and fmri : Does love last forever ?*, NeuroImage, 62 (2012), pp. 871 – 880. 20 YEARS OF fMRI.

[169] J. D. Power, *A simple but useful way to assess fMRI scan qualities*, NeuroImage, (2017), p. 9.

[170] J. D. Power, A. Mitra, T. O. Laumann, A. Z. Snyder, B. L. Schlaggar, and S. E. Petersen, *Methods to detect, characterize, and remove motion artifact in resting state fMRI*, NeuroImage, (2014).

[171] `https://lecrabeinfo.net/le-role-des-processeurs-et-de-leurs-coeurs.html#quest-ce-quun-processeur`.

[172] R. H. Pruim, M. Mennes, J. K. Buitelaar, and C. F. Beckmann, *Evaluation of ICA-AROMA and alternative strategies for motion artefact removal in resting state fMRI*, NeuroImage, (2015), pp. 278–287.

[173] R. H. Pruim, M. Mennes, D. van Rooij, A. Llera, jan K. Buitelaar, and C. F. Beckmann, *ICA-AROMA : A robust ICA-based strategy for removing motion artifact from fMRI data*, NeuroImage, (2015), pp. 267–277.

[174] M. E. Raichle, *Two views of brain function*, Trends in Cognitive Sciences, 14 (2010), pp. 180 – 190.

[175] E. M. Rasmussen and P. Willett, *Efficiency of Hierarchic Agglomerative Clustering Using USING the ICL Distributed Array Processor*, Journal of Documentation, 45 (1989), pp. 1–24.

[176] P. J. Rousseeuw, *Silhouettes : A graphical aid to the interpretation and validation of cluster analysis*, Journal of Computational and Applied Mathematics, 20 (1987), pp. 53 – 65.

[177] C. S. Roy and C. S. Sherrington, *On the regulation of the blood-supply of the brain*, (1890).

[178] R. S. Patel, D. Borsook, and L. Becerra, *Modulation of resting state functional connectivity of the brain by naloxone infusion*, 2 (2008), pp. 11–20.

[179] F. K. A. Salem and M. A. Arab, *Cache-oblivious matrix multiplication for exact factorisation*, CoRR, abs/1705.04807 (2017).

[180] A. Sarje and S. Aluru, *All-pairs computations on many-core graphics processors*, Parallel Computing, 39 (2013), pp. 79–93.

[181] `https://support.sas.com/documentation/cdl/en/statug/63347/HTML/default/viewer.htm#statug_cluster_sect012.htm`.

[182] V. Schmithorst and S. K. Holland, *Multiple Networks Recruited during a Story Processing Task Found using Group Inferences across Subjects from Independent Component Analysis*, (2010).

[183] V. J. Schmithorst and S. K. Holland, *Comparison of three methods for generating group statistical inferences from independent component analysis of functional magnetic resonance imaging data*, Journal of Magnetic Resonance Imaging, 19 (2004), pp. 365–368.

[184] E. Seifritz, F. Esposito, F. Hennel, H. Mustovic, J. G. Neuhoff, D. Bilecen, G. Tedeschi, K. Scheffler, and F. D. Salle, *Spatiotemporal Pattern of Neural Processing in the Human Auditory Cortex*, Science Mag, (2002).

[185] C. Shalizi, *Distances between Clustering, Hierarchical Clustering.* `http://www.stat.cmu.edu/~cshalizi/350/lectures/08/lecture-08.pdf`, September 2009.

[186] J. S. Siegel, J. D. Power, J. W. Dubis, A. C. Vogel, J. A. Church, B. L. Schlaggar, and S. E. Petersen, *Statistical improvements in functional magnetic resonance imaging analyses produced by censoring high-motion data points*, Human Brain Mapping, 35 (2013), pp. 1981–1996.

[187] J. S. Siegel, J. D. Power, J. W. Dubis, A. C. Vogel, J. A. Church, B. L. Schlaggar, and S. E. Petersen, *Statistical improvements in functional magnetic resonance imaging analyses produced by censoring high-motion data points*, Human Brain Mapping, (2014).

[188] S. M. Smith, A. Hyvärinen, G. Varoquaux, K. L. Miller, and C. F. Beckmann, *Group-PCA for very large fMRI datasets*, NeuroImage, 101 (2014), pp. 738–749.

[189] J. Soch and C. Allefeld, *MACS – a new SPM toolbox for model assessment, comparison and selection*, Journal of Neuroscience Methods, 306 (2018), pp. 19–31.

[190] J. Soch, J.-D. Haynes, and C. Allefeld, *How to avoid mismodelling in GLM-based fMRI data analysis : cross-validated Bayesian model selection*, NeuroImage, 141 (2016), pp. 469–489.

[191] D. A. Soltysik, D. Thomasson, S. Rajan, and N. Biassou, *Improving the use of principal component analysis to reduce physiological noise and motion artifacts to increase the sensitivity of task-based fMRI*, Journal of Neuroscience Methods, 241 (2015), pp. 18–29.

[192] C. Sorg, V. Riedl, M. Mühlau, V. D. Calhoun, T. Eichele, L. Läer, A. Drzezga, H. Förstl, A. Kurz, C. Zimmer, and A. M. Wohlschläger, *Selective changes of resting-state networks in individuals at risk for alzheimer's disease*, Proceedings of the National Academy of Sciences, 104 (2007), pp. 18760–18765.

[193] C. Spence, *Multisensory Flavor Perception*, Cell, 161 (2015), pp. 24–35.

[194] `https://stats.stackexchange.com/questions/195446/choosing-the-right-linkage-method-for-hierarchical-clustering?utm_medium=organic&utm_source=google_rich_qa&utm_campaign=google_rich_qa`.

[195] S. Sten, K. Lundengård, S. Witt, G. Cedersund, F. Elinder, and M. Engström, *Neural inhibition can explain negative bold responses : A mechanistic modelling and fmri study*, NeuroImage, 158 (2017), pp. 219 – 231.

[196] S. Suzuki, L. Cross, and J. O'Doherty, *Elucidating the underlying components of food valuation in the human orbitofrontal cortex*, 20 (2017).

[197] M. Svensén, F. Kruggel, and H. Benali, *ICA of fMRI Group Study Data*, NeuroImage, 16 (2002).

[198] `http://mriquestions.com/t2-vs-t2.html`.

[199] D. Tang, L. Fellows, D. Small, and A. Dagher, *Food and drug cues activate similar brain regions : A meta-analysis of functional MRI studies*, Physiology & Behavior, 106 (2012), pp. 317–324.

[200] M. Tchiboukdjian, *Algorithmes parallèles efficaces en cache Applications à la visualisation scientifique*, PhD thesis, Université de Grenoble, 2010.

[201] B. Thirion, G. Flandin, P. Pinel, A. Roche, P. Ciuciu, and J.-B. Poline, *Dealing with the shortcomings of spatial normalization : Multi-subject parcellation of fMRI datasets*, Human Brain Mapping, 27 (2006), pp. 678–693.

[202] B. Thirion, G. Varoquaux, E. Dohmatob, and J.-B. Poline, *Which fMRI clustering gives good brain parcellations ?*, Frontiers in Neuroscience, 8 (2014).

[203] C. G. Thomas, R. A. Harshman, and R. S. Menon, *Noise reduction in BOLD-Based fMRI Using Component Analysis*, NeuroImage, (2002).

[204] T. Thomas-Danguin, C. Rouby, G. Sicard, M. Vigouroux, V. Farget, A. Johanson, A. Bengtzon, G. Hall, W. Ormel, C. Graaf, F. Rousseau, and J.-P. Dumont, *Development of the etoc : A european test of olfactory capabilities*, 41 (2003), pp. 142–51.

[205] B. T. Thomas Yeo, F. M. Krienen, J. Sepulcre, M. R. Sabuncu, D. Lashkari, M. Hollinshead, J. L. Roffman, J. W. Smoller, L. Zöllei, J. R. Polimeni, B. Fischl, H. Liu, and R. L. Buckner, *The organization of the human cerebral cortex estimated by intrinsic functional connectivity*, Journal of Neurophysiology, 106 (2011), pp. 1125–1165.

[206] J. Tohka, *Non-parametric test for inter-subject correlations*, p. 3.

[207] J. Tohka, K. Foerde, A. R. Aron, S. M. Tom, A. W. Toga, and R. A. Poldrack, *Automatic independent component labeling for artifact removing in fMRI*, NeuroImage, (2008).

[208] B. O. Turner, E. J. Paul, M. B. Miller, and A. K. Barbey, *Small sample sizes reduce the replicability of task-based fMRI studies*, Communications Biology, 1 (2018).

[209] R. Turner, A. Howseman, G. E. Rees, O. Josephs, and K. Friston, *Functional magnetic resonance imaging of the human brain : data acquisition and analysis*, Experimental Brain Research, 123 (1998), pp. 5–12.

[210] K. Uludağ and P. Blinder, *Linking brain vascular physiology to hemodynamic response in ultra-high field mri*, NeuroImage, 168 (2018), pp. 279 – 295. Neuroimaging with Ultra-high Field MRI : Present and Future.

[211] J. E. (URCA) and G. R. (NVIDIA), *Asynchronisme*. REIMS GPU SPRING SCHOOL 2016, 2016.

[212] M. van den Heuvel, R. Mandl, and H. Hulshoff Pol, *Normalized cut group clustering of resting-state fmri data*, PLOS ONE, 3 (2008), pp. 1–11.

[213] L. van der Laan, D. de Ridder, M. Viergever, and P. Smeets, *The first taste is always with the eyes : A meta-analysis on the neural correlates of processing visual food cues*, NeuroImage, 55 (2011), pp. 296–303.

[214] F. van Meer, L. N. van der Laan, R. A. Adan, M. A. Viergever, and P. A. Smeets, *What you see is what you eat : An ALE meta-analysis of the neural correlates of food viewing in children and adolescents*, NeuroImage, 104 (2015), pp. 35–43.

[215] M. G. Veldhuizen, G. Bender, R. T. Constable, and D. M. Small, *Trying to detect taste in a tasteless solution : Modulation of early gustatory cortex by attention to taste*, Chemical Senses, 32 (2007), pp. 569–581.

[216] R. Viviani, G. Grön, and M. Spitzer, *Functional principal component analysis of fMRI data : Functional PCA of fMRI Data*, Human Brain Mapping, 24 (2005), pp. 109–129.

[217] P. Vuilleumier and J. Driver, *Modulation of visual processing by attention and emotion : windows on causal interactions between human brain regions*, Philosophical Transactions of the Royal Society B : Biological Sciences, 362 (2007), pp. 837–855.

[218] T. D. Wager, M. Lindquist, and L. Kaplan, *Meta-analysis of functional neuroimaging data : current and future directions*, Social Cognitive and Affective Neuroscience, 2 (2007), pp. 150–158.

[219] F. Wang, H.-H. Franco-Penya, J. D. Kelleher, J. Pugh, and R. Ross, *An Analysis of the Application of Simplified Silhouette to the Evaluation of k-means Clustering Validity*, in Machine Learning and Data Mining in Pattern Recognition, P. Perner, ed., vol. 10358, Springer International Publishing, Cham, 2017, pp. 291–305.

[220] Y. Wang and T.-Q. Li, *Analysis of Whole-Brain Resting-State fMRI Data Using Hierarchical Clustering Approach*, PLoS ONE, 8 (2013), p. e76315.

[221] Z. Wang, M. Xia, Z. Jin, L. Yao, and Z. Long, *Temporally and Spatially Constrained ICA of fMRI data Analysis*, PLOSOne, 9 (2014).

[222] N. Weiskopf, C. Hutton, O. Josephs, R. Turner, and R. Deichmann, *Optimized epi for fmri studies of the orbitofrontal cortex : compensation of susceptibility-induced gradients in the readout direction*, Magnetic Resonance Materials in Physics, Biology and Medicine, 20 (2007), p. 39.

[223] Wikipedia, *Article de wikipedia sur la valeur p.* `http://en.wikipedia.org/wiki/P-value`.

[224] S. M. Wilson, I. Molnar-Szakacs, and M. Iacoboni, *Beyond Superior Temporal Cortex : Intersubject Correlations in Narrative Speach Comprehension*, Cerebral Cortex, 18 (2008).

[225] D. Wishart, *An algorithm for hierarchical classifications.*, Biometrics, 25 (1969), pp. 165–170.

[226] S. Wohl, *The Experience of Eating*, Apr. 2011.

[227] C. Wolf, *Calcul parallèle ! et GPU !* `https://perso.liris.cnrs.fr/cwolf/teaching/gpu/sem5if-gpu-seance1.pdf#page=45&zoom=auto,-122,218`.

[228] Y. Xu, Y. Tong, S. Liu, H. M. Chow, N. Y. AbdulSabur, G. S. Mattay, and A. R. Brauna, *Denoising the speaking brain : Toward a robust technique for correcting artifact-contaminated fMRI data under severe motion*, NeuroImage, 103 (2014).

[229] S.-H. Yee and J.-H. Gao, *Improved detection of time windows of brain responses in fMRI using modified temporal clustering analysis*, Magnetic resonance imaging, 20 (2002), pp. 17–26.

[230] A. W. K. Yeung, T. K. Goto, and W. K. Leung, *Basic taste processing recruits bilateral anteroventral and middle dorsal insulae : An activation likelihood estimation meta-analysis of fMRI studies*, Brain and Behavior, 7 (2017), p. e00655.

[231] A. W. K. Yeung, T. K. Goto, and W. K. Leung, *Affective value, intensity and quality of liquid tastants/food discernment in the human brain : An activation likelihood estimation meta-analysis*, NeuroImage, 169 (2018), pp. 189–199.

[232] W. Zeng, A. Qiu, B. Chodkowski, and J. J. Pekar, *Spatial and temporal reproductibility-based ranking of the independant components of BOLD fMRI data*, NeuroImage, (2009).

[233] Q. Zhang and Y. Zhang, *Hierarchical clustering of gene expression profiles with graphics hardware acceleration*, Pattern Recognition Letters, 27 (2006), pp. 676–681.

[234] X. Zhao, D. Glahn, L. H. Tan, N. Li, J. Xiong, and J.-H. Gao, *Comparison of TCA and ICA techniques in fMRI data processing*, Journal of Magnetic Resonance Imaging, 19 (2004), pp. 397–402.

Index

K

M

N

P

R

S

T

Glossaire

B

B-spline Combinaison linéaire de splines (fonction définie par morceaux par des polynômes) positive.

C

Classifieur La tâche d'un classifieur consiste à attribuer une classe à une forme inconnue qui lui est présentée ou à refuser de lui attribuer une classe si la forme inconnue est trop éloignée de celles connues.

CUDA Langage de programmation pour les GPUs NVIDIA.

F

Flops Nombre d'opérations flottantes effectuées en une seconde (1 GFlops=10^9 opérations flottantes par seconde).

Fonction objectif Fonction qui sert de critère pour déterminer la meilleure solution à un problème d'optimisation mathématique

H

Hétéroscédasticité En statistique, on parle d'hétéroscédasticité lorsque les variances des résidus des variables examinées sont différentes. *Elle s'oppose à homoscédasticité.*

Homoscédasticité On parle d'homoscédasticité lorsque la variance des erreurs stochastiques de la régression est la même pour chaque observation.

M

MNI C'est un des espaces de référence des coordonnées cérébrales crée par le *Montreal Neurological Institute*.

MPI Interface de programmation pour générer un programme par échange de messages adapté aux architecture à mémoire distribuée.

O

Onset Signal créneau valant 1 lors de la simulation 0 sinon.

OpenMP Interface de programmation utilisant les processus légers pour les architectures à mémoire partagée.

Opération flottante Les opérations en virgule flottante (additions ou multiplications) incluent toutes les opérations qui impliquent des nombres réels

P

Puissance statistique La puissance d'un test est la probabilité de rejeter l'hypothèse nulle à raison c'est-à-dire lorsqu'on est dans le cadre de l'hypothèse alternative.

S

Scan Image cérébrale 3D

SNR En imagerie, le ratio signal sur bruit est défini par le rapport de la moyenne de la valeur du signal sur l'écart-type des valeurs du signal.

Socket Connecteur utilisé pour interfacer un processeur avec une carte mère.

T

Taille de l'effet La taille de l'effet mesure de combine on s'écarte de l'hypothèse nulle. Plus la taille de l'effet est grande, plus il est justifié de rejeter l'hypothèse nulle.

Test non-paramétrique Les tests non-paramétriques ne se fondent pas sur des distributions statistiques. Ils peuvent donc être utilisés même si les conditions de validité des tests paramétriques ne sont pas vérifiées.

Transformation z de Fisher La transformation z de Fisher permet de convertir le coefficient de corrélation r en une variable z suivant une loi normale :

$$z = \frac{1}{2} \ln \left(\frac{1+r}{1-r} \right) \quad .$$

V

Voxel Équivalent d'un pixel pour une image 3D