UNIVERSITÉ DU QUÉBEC À MONTRÉAL

ALGORITHMES POUR LA DÉTECTION DE TRANSFERTS HORIZONTAUX DE GÈNES COMPLETS ET PARTIELS

THÈSE PRÉSENTÉE

COMME EXIGENCE PARTIELLE

DU DOCTORAT EN INFORMATIQUE

PAR

ALPHA BOUBACAR DIALLO

DÉCEMBRE 2012

À mes parents, Alpha Oumar Diallo et Aïssatou Damba Diallo.

REMERCIEMENTS

Je tiens à témoigner toute ma reconnaissance à mon directeur de recherche Vladimir Makarenkov, professeur au département d'informatique à l'Université du Québec à Montréal (UQAM), pour le soutien qu'il m'a apporté pendant toute la durée de mon doctorat. Il a toujours été là pour m'aider à mener à bien ce travail à travers son suivi, ses conseils et ses suggestions. Qu'il trouve ici l'expression de ma gratitude et de ma profonde reconnaissance.

Je souhaite aussi remercier les membres du laboratoire de recherche en bioinformatique de l'UQAM, mes collègues Dunarel Badescu, Alix Boc, Mikael Leclercq, Etienne Lord, Nadia Tahiri, ainsi que le professeur Abdoulaye Baniré Diallo, pour leurs points de vue et leur soutien qui ont été d'une grande utilité et, surtout, pour avoir contribué à la création d'une ambiance de travail propice.

Je remercie sincèrement mes évaluateurs, Wessam Ajib, Farid Beninel, François Brucker, Roger Nkambou et Pedro Peres-Neto pour leurs corrections et suggestions constructives qui m'ont permis d'améliorer ma thèse de doctorat de façon significative.

Je tiens aussi à remercier Dr. Étienne Gagnon, professeur et directeur des programmes maîtrise et de doctorat en informatique à l'UQAM pour l'aide fournie pour une meilleure compréhension de l'environnement de développement de compilateurs et d'interpréteurs *SableCC*.

J'adresse ma totale reconnaissance aux Fonds Québécois de la Recherche sur la Nature et les Technologies (FQRNT) ainsi qu'à la Fondation de l'Université du Québec à Montréal qui ont contribué au financement de ce projet.

Je ne pourrais finir cette liste sans dire un grand merci aux membres de ma famille qui m'ont soutenu moralement et financièrement pendant ces années et sans lesquels la réalisation de ce travail n'aurait pas été possible.

À mes amis et à tous ceux qui ont contribué de près ou de loin à la réalisation de ce projet, qu'ils trouvent ici mes remerciements les plus sincères.

TABLE DES MATIÈRES

LISTE DES FIGURES

LISTE DES TABLEAUX

LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES

ADN	Acide Désoxyribonucléique
ALU	Arithmetic Logic Unit
ANSI	American National Standards Institute
API	Application Programming Interface
ARN	Acide Ribonucléique
ASM	Alignement de Séquences Multiples
BLAST	Basic Local Alignment Search Tool
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
DRAM	Dynamic Random Access Memory
FLOP	FLoating point Operations Per Second
GB	Gigabyte
GCD	Greatest Common Divisor
GPU	Graphics Processing Unit
HGT / THG	Horizontal Gene Transfer / Tranfert Horizontal de Gène(s)
ISO	International Organization for Standardization
LVI	Linear Variable Interval
MC	Moindres Carrés
MIMD	Multiple Instruction Multiple Data Stream
MISD	Multiple Instruction Single Data Stream
ML	Maximum Likelihood
MMS	Massingill, Mattson et Sanders
MPI	Message Passing Interface
NJ	Neighbor-Joining
OpenCL	Open Computing Language
OpenGL	Open Graphics Library

OpenMP	Open Multi-Processing
PC	Personal Computer
PHYLIP	PHYLogeny Inference Package
PhyML	Phylogenetic estimation using Maximum Likelihood
PLATO	Programming Languages Analysis Translation and Optimization
PVM	Parallel Virtual Machine
PVI	Polynomial Variable Interval
RF	Robinson et Foulds
RVI	Rational Variable Interval
SIMD	Single Instruction Multiple Data Stream
SISD	Single Instruction Single Data Stream
SPMD	Single Process Multiple Data Stream / Single Program Multiple Data Stream
SPR	Subtree Prune and Regraft
SUIF	Stanford University Intermediate Format
TCP/IP	Transmission Control Protocol / Internet Protocol
UPGMA	Unweighted Pair-Group Method using arithmetic Averages
URL	Uniform Resource Locator
VFC	Vienna Fortran compiler
VGA	Video Graphics Array

RÉSUMÉ

Avec l'arrivée des données moléculaires vers la fin des années 70, nous avons assisté à la découverte de nouveaux mécanismes d'évolution primordiaux dont l'échange du matériel génétique entre les espèces. Un tel échange peut se faire horizontalement, quand l'organisme intègre le matériel génétique provenant d'un autre organisme qui n'est pas son descendant direct, ou verticalement, quand l'organisme reçoit du matériel génétique à partir de son ancêtre le plus proche. Le problème de la détection et de la classification des transferts horizontaux de gènes (THG) est parmi les plus ardus en bioinformatique. Dans cette thèse, nous décrivons cinq nouveaux algorithmes pour la détection de THGs complets ou partiels qui seront basés sur des comparaisons topologiques et métriques entre un arbre d'espèces et un arbre de gène inférés pour le même ensemble d'espèces. Ces algorithmes incluent l'algorithme de détection de THGs complets ainsi que ses versions interactive et consensus. Les deux algorithmes de détection de transferts partiels que nous avons proposés peuvent être vus comme une généralisation de l'algorithme de détection de transferts complets. Ils peuvent être utilisés pour identifier des gènes mosaïques. Nous présentons aussi dans cette thèse une version parallèle de l'algorithme de détection de THGs complets, ainsi qu'une plateforme pour la transformation semi-automatique de programmes bioinformatiques séquentiels en programmes parallèles. Une interface Web intégrant tous les programmes développés dans le cadre de ce projet doctoral a aussi été mise au point.

Mots clés : algorithmes bioinformatiques, arbre phylogénétique, programmation parallèle, réseau réticulé, transfert horizontal de gènes (THG).

INTRODUCTION

Cette thèse porte sur le développement de nouveaux modèles et algorithmes pour la détection de transferts horizontaux de gènes (THGs) *complets* et *partiels*, de même que sur la parallélisation de ces algorithmes.

Le transfert horizontal de gènes (THG) est un mécanisme d'évolution important qui permet à des organismes, surtout à des bactéries et à des virus, de s'échanger des gènes (Koonin, 2003). C'est un processus au cours duquel un organisme intègre le matériel génétique provenant d'un autre organisme qui n'est pas son descendant direct (Nelson *et al.*, 1999). Contrairement au transfert horizontal, le transfert vertical se produit lorsque l'organisme reçoit du matériel génétique à partir de son ancêtre le plus proche. Les différentes techniques existantes pour la détection de THGs sont basées principalement sur l'analyse de la composition nucléotidique des séquences biologiques et sur l'étude des incongruences topologiques entre les arbres phylogénétiques qui permettent aux biologistes d'émettre une hypothèse de transfert horizontal de gènes (Hein, 1993; von Haeseler et Churchill, 1993; Mirkin *et al.*, 1995; Makarenkov *et al.*, 2008; Boc *et al.*, 2010).

Pour mieux comprendre la problématique du transfert horizontal de gènes abordée dans cette thèse, nous introduisons d'abord dans le chapitre I les définitions de base en bioinformatique et présentons le processus de reconstruction d'arbres phylogénétiques et de réseaux réticulés. Nous nous attardons particulièrement sur le mécanisme du THG qui représente un volet important de l'évolution réticulée.

Dans le chapitre II, nous présentons une méthode de détection de transferts horizontaux de gènes *complets* basée sur l'algorithme que nous avons proposé dans Makarenkov, Boc et Diallo Al. (2007). Nous décrirons aussi une version plus complète de cette méthode présentée dans Boc *et al.* (2010) avant d'introduire deux nouveaux algorithmes pour la construction d'un scénario de consensus de THGs et pour une validation interactive des transferts détectés.

Dans le chapitre III, nous présentons deux nouveaux algorithmes pour la détection de transferts horizontaux de gènes *partiels* menant à la formation des gènes mosaïques (Makarenkov, Boc, Diallo Al. et Diallo Ab., 2008; Boc, Diallo Al. et Makarenkov, 2011).

Dans le chapitre IV, nous introduisons les définitions et les notions de base de la programmation parallèle. Ensuite, nous présentons une plateforme pour la transformation semi-automatique de programmes bioinformatiques séquentiels en programmes parallèles en utilisant la bibliothèque standard d'échange de messages, MPI (Message Passing Interface). Nous terminerons ce chapitre par la présentation de la version parallèle de l'algorithme de détection de THGs *complets* ainsi que des résultats des simulations exposant les gains en performance obtenus. Un article (Diallo Al., Lord et Makarenkov, 2012) portant sur l'interface de traduction semi-automatique a été soumis aux actes d'une conférence internationale.

Le chapitre V présente les interfaces Web développées pour les différents programmes de détection de transferts horizontaux de gènes, incluant de nouvelles interfaces pour une détection interactive de transferts horizontaux et le calcul d'un scénario de consensus de ces derniers. Un article (Boc, Diallo Al. et Makarenkov, 2012), portant sur l'interface Web que nous avons développée, a été accepté pour publication dans la prestigieuse revue *Nucleic Acids Research* (Web Server Issue).

En conclusion, nous faisons une synthèse du travail effectué et abordons quelques pistes possibles pour l'amélioration des résultats obtenus.

Dans le cadre de cette thèse, nous nous fixons trois objectifs. Le premier objectif consiste à développer et à mettre en place des algorithmes pour la détection de transferts horizontaux de gènes complets : algorithme standard, algorithme de consensus et algorithme interactif. Ces algorithmes seront basés sur la réconciliation des phylogénies d'espèces et de gène, tout en considérant des règles d'évolution fondamentales. Le second objectif consiste à développer et à mettre en place deux nouveaux algorithmes pour la détection de transferts horizontaux de gènes partiels permettant l'identification des gènes mosaïques. Le troisième objectif consiste au développement d'interfaces Web de tous les programmes de détection de THGs complets et partiels que nous avons proposés et à l'implémentation d'une version parallèle du programme de détection de THGs complets.

CHAPITRE I

BIOINFORMATIQUE ET PHYLOGENÈSE

Dans ce chapitre, nous introduisons des définitions et des notions de base de la bioinformatique, et plus précisément de la phylogenèse. Ainsi, nous présenterons le processus et les différentes méthodes de reconstruction d'arbres phylogénétiques et de réseaux réticulés. Nous nous attarderons particulièrement sur le mécanisme du transfert horizontal de gènes qui représente un volet important de l'évolution réticulée.

1.1 Phylogenèse

La phylogenèse, ou phylogénie, est définie comme « *l'histoire de la formation et de l'évolution d'une espèce* ». Le terme phylogenèse, introduit par Haeckel (1874), provient du grec *phûlon*, « *tribu* », et *genesis* « *origine* ». Haeckel l'a définie comme « *l'histoire du développement paléontologique des organismes par analogie avec l'ontogénie ou histoire du développement individuel* ». La phylogénie constitue une théorie servant à construire des classifications d'espèces. La phylogénie moléculaire, dont certaines méthodes de reconstruction seront présentées par la suite, étudie l'histoire évolutive des espèces en se basant sur une portion de leur séquence génomique.

Un arbre est dit arbre phylogénétique (voir la Figure 1.1), appelé aussi souvent phylogénie, si le concept de descendance des espèces avec modification des sites des séquences des espèces a été utilisé lors de son inférence, c'est-à-dire sa reconstruction (Barthélémy et Guénoche, 1991). Un arbre phylogénétique présente les relations de parenté entre les organismes vivants. Il représente ainsi les relations phylogénétiques entre les espèces étudiées. L'arbre est une forme de classification des espèces.

Un arbre phylogénétique est composé de quatre éléments principaux (voir la Figure 1.1) :

- la racine, qui indique l'ancêtre commun des espèces représentées dans l'arbre : les arbres phylogénétiques peuvent être enracinés ou non-enracinés;
- les nœuds externes ou les feuilles, qui représentent les espèces contemporaines dont les informations ont été utilisées lors de la construction de l'arbre;
- les nœuds internes, qui représentent des ancêtres hypothétiques;
- les branches ou arêtes, qui montrent les relations de descendance entre les nœuds de l'arbre.

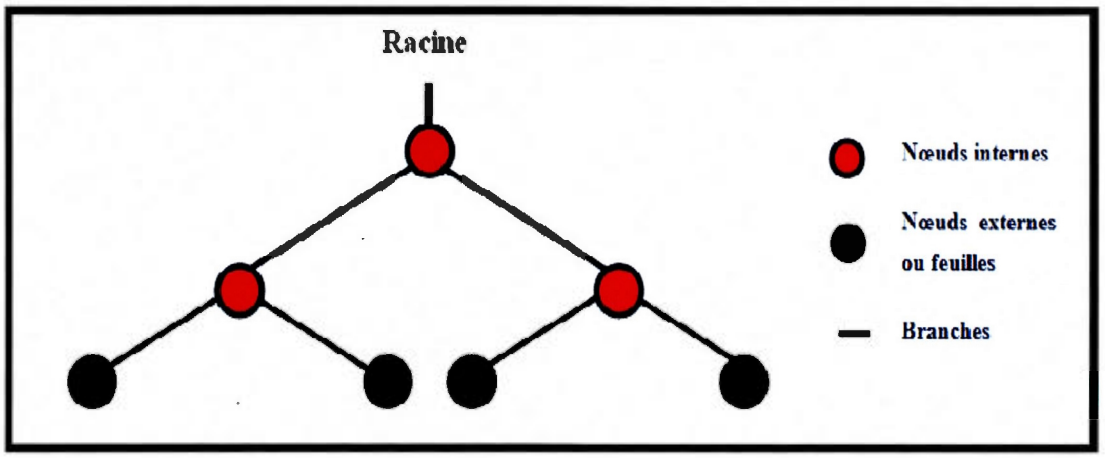


Figure 1.1 - Un arbre phylogénétique.

Le degré d'un nœud de l'arbre est défini comme le nombre de branches adjacentes à ce nœud. Tout nœud de degré supérieur à trois est appelé « non résolu », sinon le nœud est résolu. Un arbre phylogénétique binaire *non-enraciné* ayant n espèces (feuilles) dont tous les nœuds internes sont résolus est composé de :

- $2n$-2 nœuds (n-2 nœuds internes et n feuilles);
- $2n$-3 branches.

1.2 Théories de l'évolution des espèces

En 1858, Darwin (1858) a introduit la notion d'arbre phylogénétique à travers sa théorie d'évolution par sélection. Ses travaux ont contribué grandement à l'utilisation d'arbres comme support formel de la représentation des relations entre les espèces. Ce support est devenu un élément incontournable pour leur classification. Huit ans plus tard, Haeckel (1866) présenta le premier arbre universel du vivant (voir la

Figure 1.2), incluant la plupart des espèces et tous les groupes connus à l'époque. Étrangement, cet arbre fut remarquablement correct parce qu'il ressemble en plusieurs éléments à ce que l'on connaît actuellement de l'évolution des organismes, sauf en ce qui concerne les microorganismes et, en particulier, les procaryotes et les virus. Au cours des décennies suivantes, les seuls moyens de classification des espèces étaient basés sur des comparaisons entre leur morphologie, leur comportement et leur répartition géographique.

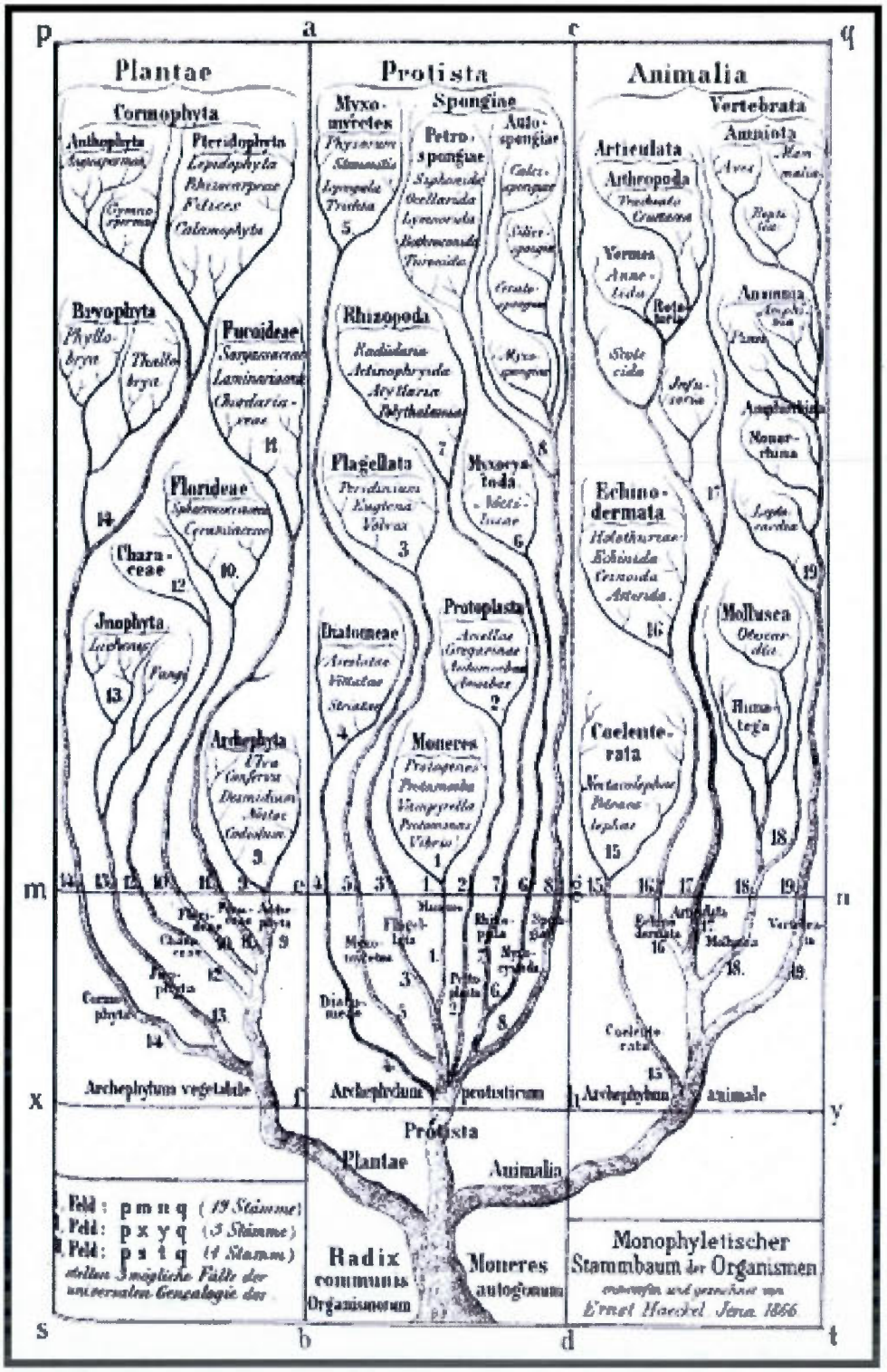


Figure 1.2 - Arbre d'évolution de Haeckel (1866).

Il a fallu attendre l'arrivée des données moléculaires, vers la fin des années 70 et au début des années 80, pour assister à la découverte de nouveaux mécanismes d'évolution importants tels que l'échange du matériel génétique entre des organismes appartenant à différentes espèces (Doolittle, 1999). Actuellement, les arbres phylogénétiques peuvent être inférés à partir d'un ensemble varié de données, tels que les caractères discrets, les séquences moléculaires, les fréquences des gènes, les sites de restriction, l'ordre des gènes, les microsatellites, etc.

La manière classique d'illustrer les relations phylogénétiques entres les espèces est de les modéliser en utilisant un arbre phylogénétique. La reconstruction d'un arbre phylogénétique débute par ce que l'on appelle l'alignement qui consiste à mettre en correspondance les sites des séquences des espèces de manière à pouvoir les comparer les unes aux autres. Les séquences utilisées dans la procédure de reconstruction peuvent être des séquences d'ADN, des séquences d'ARN ou des séquences d'acides aminés (protéines).

Toutefois, plusieurs importants mécanismes phylogénétiques ne peuvent être expliqués que par le phénomène de l'évolution réticulée, qui suppose des liens supplémentaires entre les espèces par rapport au modèle d'arbre phylogénétique classique (Doolittle, 1999). L'évolution réticulée représente la part de l'évolution des espèces qui ne peut être décrite par le modèle de bifurcation traditionnel utilisé en analyse phylogénétique.

1.3 Reconstruction d'arbres phylogénétiques

Il existe quatre approches principales pour l'inférence, ou la reconstruction, de phylogénies (voir la Figure 1.3). La première, appelée approche phénétique, ne tient pas compte des relations historiques entre les espèces (Kidd et Sgaramella-Zonta, 1971; Sneath et Sokal 1973; Beyer *et al.*, 1974; Saitou et Nei, 1987; Rzhetsky et Nei, 1992; Makarenkov et Leclerc, 1999). Elle est basée sur le calcul de distances, ou dissimilarités, entre les espèces à partir des séquences alignées (Cavalli-Sforza et Edwards, 1967; Fitch et Margoliash, 1967), puis sur la reconstruction d'un arbre phylogénétique, à partir de ces distances, en utilisant une procédure de regroupement (souvent hiérarchique). La deuxième approche, appelée approche de maximum de parcimonie (Edwards et Cavalli-Sforza, 1963; Farris, 1970; Fitch, 1971),

considère plusieurs pistes de l'évolution, en inférant une séquence d'ancêtre au niveau de chaque nœud et en choisissant un arbre optimal selon le modèle d'évolution sélectionné. La troisième approche est l'approche probabiliste ou le maximum de vraisemblance (Edwards et Cavalli-Sforza, 1964; Neyman 1971; Felsenstein, 1981; Guindon et Gascuel, 2003). Elle évalue, en termes de probabilités, l'ordre des branchements et la longueur des arêtes d'un arbre sous un modèle évolutif donné. Les méthodes Bayésiennes (Rannala et Yang, 1996; Rannala, 1997; Li *et al.*, 2000; Huelsenbeck et Ronquist, 2001) forment la dernière approche pour l'inférence ou la reconstruction de phylogénies.

Nous présenterons, sur la Figure 1.3, les quatre principales approches de reconstruction d'arbres phylogénétiques : l'approche basée sur les distances (approche phénétique), l'approche basée sur le maximum de parcimonie, l'approche basée sur le maximum de vraisemblance et l'approche Bayésienne. Les trois dernières approches appartiennent à l'approche cladistique.

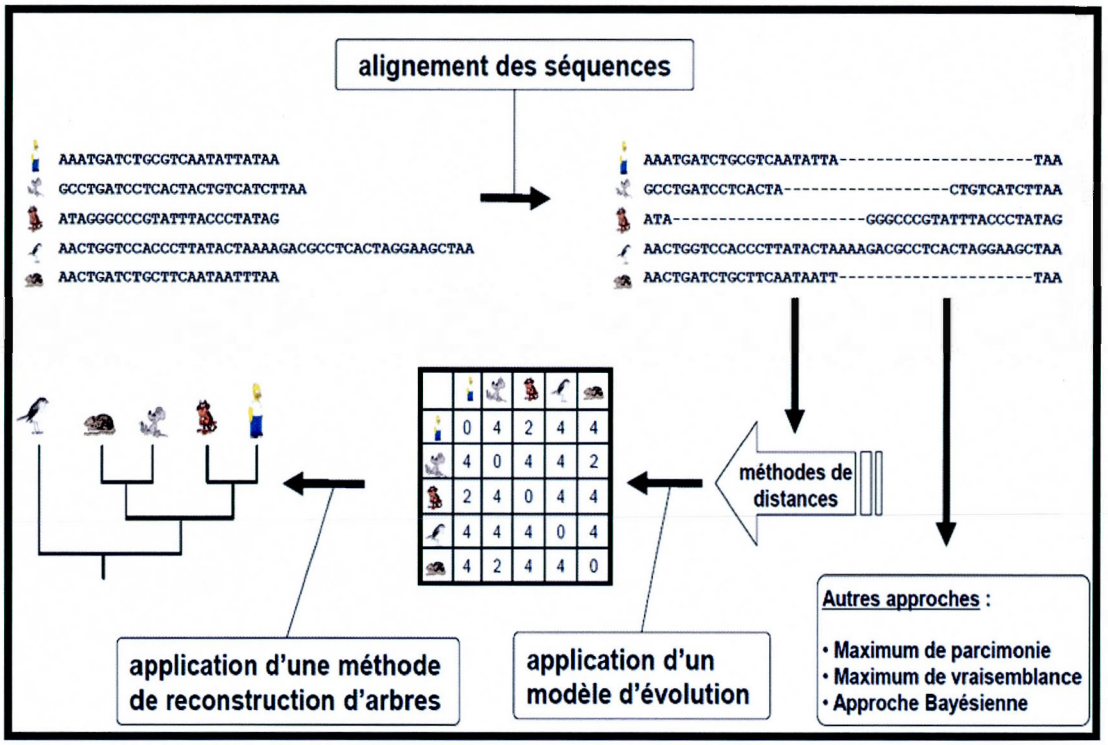


Figure 1.3 - Processus d'inférence d'arbres phylogénétiques à travers les quatre approches existantes.

1.3.1 Approche basée sur les distances

Cette approche d'inférence d'arbres phylogénétiques a été introduite par Cavalli-Sforza et Edwards (1967) et Fitch et Margoliash (1967). Dans cette approche, on calcule d'abord les dissimilarités entre chaque paire d'espèces avant d'inférer l'arbre phylogénétique dont les longueurs des branches se rapprochent le mieux aux dissimilarités calculées. La somme des longueurs des branches entre deux espèces i et j de l'arbre phylogénétique est appelée distance évolutive entre les espèces i et j. La distance évolutive correspond au nombre d'évènements mutationnels réels, contrairement à la dissimilarité qui est son estimation obtenue à partir d'une comparaison des séquences alignées. La distance évolutive est également appelée distance additive et respecte les propriétés d'une distance arborée; voir Barthélémy et Guénoche (1991).

Si les distances estimées sont assez proches du nombre d'évènements évolutifs entre les espèces, une méthode de distances reconstruit généralement un arbre correct (Kim et Warnow, 1999). Cette supposition est vraie pour plusieurs modèles d'évolution de séquences biomoléculaires pour lesquels les méthodes basées sur les distances donnent des résultats suffisamment précis (Li, 1997). L'avantage principal des méthodes basées sur les distances est que leur temps d'exécution est très rapide, ce qui leur permet d'analyser de larges ensembles de données.

Les méthodes de distances utilisent une matrice de dissimilarités pour reconstituer la matrice de distances évolutives (distance d'arbre) qui peut être représentée sous forme d'un arbre unique (Barthélémy et Guénoche, 1991). Pour reconstruire l'arbre phylogénétique à partir d'une matrice de dissimilarités donnée, les méthodes de distances se servent de différentes techniques dont les plus connues sont les suivantes :

- Les méthodes d'ajustement par les moindres carrés

Ces méthodes d'ajustement (Gauss, 1811) reposent sur le calcul d'une distance arborée (*i.e.*, distance d'arbre) la plus proche de la matrice de dissimilarités donnée, en utilisant un critère dont la forme générale est la suivante :

$$MC = \sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij} (\delta_{ij} - d_{ij})^2$$

où **W** représente la matrice de poids accordés à la comparaison des paires de séquences;
n représente le nombre d'espèces;
δ_{ij} représente la distance évolutive entre les espèces i et j;
d_{ij} représente la dissimilarité donnée entre les espèces i et j.

Le critère des moindres carrés non pondérés de Cavalli-Sforza et Edwards (1967) correspond à l'initialisation de toutes les valeurs de la matrice de poids **W** à 1. Fitch et

Margoliash (1967) utilisent le même critère où $w_{ij} = \frac{1}{d_{ij}^2}$, tandis que Beyer *et al.* (1974) ont proposé de considérer le cas où $w_{ij} = \frac{1}{d_{ij}}$.

- Les méthodes d'évolution minimum

Les méthodes d'évolution minimum minimisent la longueur totale des branches de l'arbre reconstruit. Une première méthode de ce type a été proposée par Kidd et Sgaramella-Zonta (1971), et puis raffinée par Saitou et Nei (1987) et Rzhetsky et Nei (1992). L'arbre est ajusté aux données et la longueur des branches est déterminée en utilisant la méthode des moindres carrés non pondérés. Les méthodes d'évolution minimum requièrent souvent un temps de calcul similaire à celui des méthodes d'ajustement. Ce temps peut être amélioré en utilisant les méthodes de Bryant et Waddell (1998) et Makarenkov et Leclerc (1999) qui optimisent le temps de calcul des moindres carrés. Le temps de recherche de topologies d'arbres peut également être diminué en utilisant une recherche gloutonne comme celle proposée par Desper et Gascuel (2002).

- Les méthodes de regroupement

Les méthodes de regroupement n'utilisent pas un critère explicite pour trouver l'arbre qui correspond à la matrice de dissimilarités donnée. Elles utilisent un algorithme particulier traitant cette matrice de dissimilarités pour inférer directement l'arbre final. Elles peuvent être très rapides, mais elles ne garantissent pas une utilisation efficace de toutes les distances (Felsenstein, 2004). Parmi les algorithmes de reconstruction d'arbres phylogénétiques basés sur les distances, la méthode UPGMA (Unweighted Pair-Group Method using arithmetic Averages) fut l'une des premières à avoir été proposée (Sneath et Sokal, 1973). L'algorithme NJ (Neighbor-Joining) de Saitou et Nei (1987) est, de loin, le plus populaire.

L'algorithme de NJ se présente comme suit (Saitou et Nei, 1987) :

1. Pour chaque espèce *i*, calculer : $u_i = \sum_{j:j\neq i}^{n} \frac{\delta_{ij}}{n-2}$, où δ représente la valeur de la dissimilarité donnée entre les espèces *i* et *j*; l'arbre initial a une forme de buisson où les espèces sont représentées par les feuilles;
2. Choisir les espèces *i* et *j* pour lesquelles la distance $\delta_{ij} - u_i - u_j$ est la plus courte;
3. Fusionner *i* et *j* choisis à l'étape 2 et calculer les longueurs des branches menant au nœud (*vi*) et au nœud (*vj*) de la manière suivante :

$$v_i = \frac{1}{2}\delta_{ij} + \frac{1}{2}(u_i - u_j),$$
$$v_j = \frac{1}{2}\delta_{ij} + \frac{1}{2}(u_j - u_i);$$

4. Calculer les distances entre le nouveau nœud (*ij*) et les feuilles restantes en utilisant la formule suivante : $\delta_{(ij)k} = \frac{\delta_{ik} + \delta_{jk}}{2}$;
5. Supprimer les feuilles *i* et *j* de la matrice de distances et les remplacer par le nouveau nœud (*ij*) qui sera considéré comme une feuille;
6. Revenir à l'étape 1 si le nombre de nœuds restants est supérieur à 3. Sinon, connecter les trois nœuds restants par des branches.

1.3.2 Approche basée sur le maximum de parcimonie

L'idée générale du principe de maximum de parcimonie a été formulée quand la première méthode de ce type a été introduite par Edwards et Cavalli-Sforza (1963). Les méthodes de maximum de parcimonie infèrent des arbres phylogénétiques en évaluant le nombre de mutations possibles entres les séquences données. En terme général, le but de ces méthodes est de trouver l'arbre phylogénétique de longueur totale minimale, c'est-à-dire l'arbre qui a le plus petit nombre d'évènements mutationnels possibles. Ceci implique deux problématiques :

(1) La capacité d'effectuer la reconstruction impliquant le moins d'évènements possibles; (2) La capacité de chercher parmi toutes les phylogénies qui minimisent le nombre d'évènements.

Il existe plusieurs variations de la méthode de parcimonie dont les plus utilisées sont la parcimonie de Wagner (Farris, 1970) et celle de Fitch (1971). Habituellement, on retrouve plus qu'un arbre qui minimise le nombre d'évènements mutationnels en appliquant une méthode de maximum de parcimonie. Afin de garantir de retrouver le meilleur arbre, une évaluation exhaustive de toutes les topologies possibles doit être effectuée. L'approche de maximum de parcimonie reconstruit correctement un arbre phylogénétique si le rapport entre le nombre d'évènements mutationnels et la position de la séquence est petit.

Lorsque le nombre d'espèces est inférieur à dix, il est possible d'effectuer une recherche exhaustive. Dans le cas contraire, il faut se contenter des constructions heuristiques d'un arbre, pour obtenir un résultat dans un temps raisonnable.

1.3.3 Approche basée le maximum de vraisemblance

L'approche de maximum de vraisemblance a été introduite par Edwards et Cavalli-Sforza (1964) dans le cadre de l'étude des données sur la fréquence des gènes. La première application d'une méthode de maximum de vraisemblance aux séquences moléculaires a été effectuée par le statisticien Neyman (1971), et imposait des contraintes sur le taux d'évolution. L'approche de maximum de vraisemblance introduite par Felsenstein (1981) n'imposait, quant à elle, aucune contrainte par rapport à la constance du taux d'évolution entre les lignées. Elle assignait des probabilités aux évènements mutationnels, plutôt que de les compter. Cette dernière méthode compare les arbres phylogénétiques possibles sur la base de leur habileté à prédire les données observées. L'arbre qui a la plus grande probabilité de produire les séquences observées est choisi comme arbre solution.

La vraisemblance (ou le score de vraisemblance) est exprimée sous forme d'un logarithme naturel. Pour trouver l'arbre le plus vraisemblable, les bases de toutes les séquences à chaque site sont considérées séparément et le logarithme de la vraisemblance est calculé pour une topologie donnée en utilisant un modèle d'évolution prédéterminé. Ce

logarithme de la vraisemblance est cumulé sur tous les sites et sa somme est maximisée pour estimer la longueur totale des branches de l'arbre. Cette procédure est répétée pour toutes les topologies possibles et la topologie ayant la plus grande vraisemblance est retenue (Felsenstein, 1981).

De façon similaire au maximum de parcimonie, une méthode de maximum de vraisemblance reconstruit un dispositif d'ancêtres au niveau de tous les nœuds de l'arbre considéré, mais elle assigne aussi aux branches une longueur basée sur les probabilités des mutations. Pour chaque topologie d'arbre possible, les taux de substitution supposés sont variés dans le but de trouver les paramètres qui produiront la plus grande vraisemblance par rapport aux séquences observées.

Le principal obstacle dans l'utilisation des méthodes de maximum de vraisemblance est le temps de calcul. Les algorithmes qui trouvent le score de vraisemblance font des recherches dans un espace multidimensionnel de paramètres. Ceci fait en sorte que la solution des problèmes à grande échelle (> 100 séquences) est extrêmement longue à trouver. Une méthode d'estimation de maximum de vraisemblance peut être l'objet d'erreurs systématiques quand le modèle d'évolution utilisé pour évaluer la vraisemblance d'un arbre donné ne reflète pas le processus actuel d'évolution.

Il existe, cependant, une méthode de maximum de vraisemblance plus rapide que les autres, appelée PhyML (Guindon et Gascuel, 2003), qui est devenue très populaire actuellement.

1.3.4 Approche basée sur les méthodes Bayésiennes

Les méthodes Bayésiennes sont étroitement liées aux méthodes de maximum de vraisemblance. À la différence de ces dernières, les méthodes Bayésiennes utilisent une distribution *a priori* sur la quantité inférée. L'utilisation d'une distribution *a priori* permet d'interpréter le résultat comme une distribution de la quantité de données. Les méthodes Bayésiennes datent des années 1970 et l'utilisation des méthodes de Chaîne de Markov de Monte Carlo (Markov Chain Monte Carlo) leur a donné une nouvelle impulsion (Rannala et Yang, 1996; Rannala, 1997; Li *et al.*, 2000; Huelsenbeck et Ronquist, 2001). L'hypothèse

principale est que l'arbre optimal est celui qui maximise la probabilité *a posteriori*. La probabilité *a posteriori* de l'hypothèse est proportionnelle à la vraisemblance multipliée par la probabilité *a priori* de cette hypothèse. Les probabilités *a priori* des différentes hypothèses dépendent des suppositions concernant les relations phylogénétiques possibles. Dans plusieurs cas, les chercheurs n'ont aucune information à propos des distributions de la probabilité *a priori*. La manière la plus simple de résoudre ce problème est de spécifier une probabilité *a priori* uniforme, avec laquelle toutes les valeurs possibles d'un paramètre ont la même probabilité *a priori*.

Dans l'analyse Bayésienne, le résultat final ne dépend pas d'une valeur spécifique, mais de la considération de toutes les valeurs possibles des paramètres. Même s'il y'a suffisamment de données pour estimer plusieurs paramètres, les algorithmes utilisés pour trouver le score de maximum de vraisemblance, peuvent être très lents et de moins en moins fiables au fur et à mesure que le nombre de paramètres augmente. Contrairement à cela, les méthodes Bayésiennes reposent sur un algorithme qui ne cherche pas forcément à trouver le point le plus haut dans l'espace des valeurs des paramètres (optimum global). Par rapport au maximum de vraisemblance, les méthodes Bayésiennes ont l'avantage d'avoir une vitesse de calcul plus élevée et la possibilité d'incorporer des modèles d'évolution plus sophistiqués.

1.4 Évolution réticulée

Il a longtemps été supposé que l'évolution des espèces est un processus de branchement ne pouvant être représenté que par une topologie d'arbre (Doolittle, 1999). Dans ce dernier, une espèce est liée seulement à son ancêtre et à ses descendants les plus proches et toutes les autres relations entre les espèces ne peuvent être prises en compte. L'évolution réticulée reflète la partie de l'évolution des espèces qui ne peut être représentée par le modèle arborescent classique. Un réticulogramme, par exemple, peut être utilisé pour représenter l'évolution réticulée (Legendre et Makarenkov, 2002).

Pour construire un réticulogramme, la première étape consiste à reconstruire l'arbre original représentant l'évolution des espèces et la seconde consiste à ajouter les réticulations (ou branches supplémentaires) nécessaires. Les réticulations sont ajoutées entre deux

branches de l'arbre original (Makarenkov et Legendre, 2004). La Figure 1.4 montre un réseau réticulé (ou un réticulogramme). Le trait ajouté entre les arêtes 1 et 2 représente une arête de réticulation ajoutée à l'arbre original.

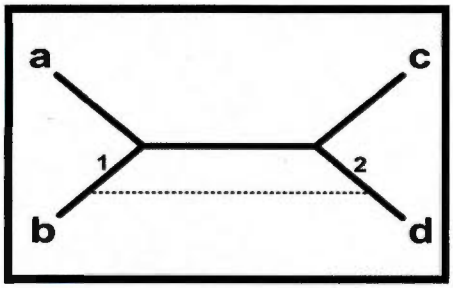


Figure 1.4 - Un réseau réticulé.

Dans son célèbre article, Doolittle (1999) a mis l'accent sur le rôle de l'évolution réticulée, et plus précisément du transfert horizontal de gènes, dans l'évolution des bactéries, de même que des espèces plus complexes. Nous pouvons remarquer dans la Figure 1.5, introduite par Doolittle, que l'évolution des espèces se produit selon un modèle en réseau plutôt qu'un modèle en arbre. Selon Doolittle, les biologistes moléculaires auraient échoué à trouver le vrai arbre de la vie non pas à cause des méthodes arborescentes traditionnelles qu'ils ont utilisées, mais parce que l'histoire de la vie ne peut être représentée correctement par un arbre. Des phénomènes très importants, tels que le transfert horizontal de gènes, l'hybridation et l'allopolyploïdie ne correspondent pas au modèle d'évolution arborescente (Legendre, 2000; Lapointe, 2000).

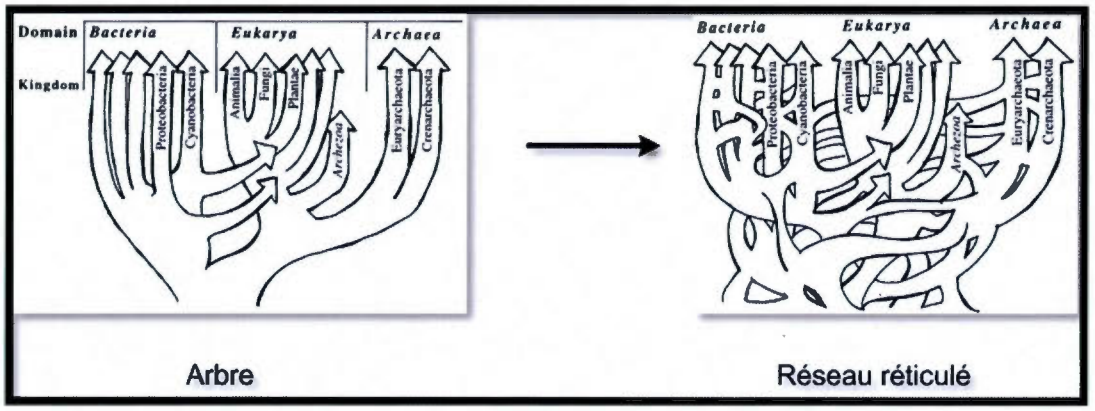


Figure 1.5 - Un réseau réticulé représente mieux l'histoire de la vie qu'un arbre phylogénétique classique (selon Doolittle, 1999).

L'évolution réticulée a longtemps été délaissée dans les analyses phylogénétiques. Les premières méthodes n'ont vu le jour qu'à partir du milieu des années 70 (Sneath *et al.*, 1975; Sonea et Panisset, 1976). Il existe plusieurs méthodes pour identifier l'évolution réticulée dans les séquences de nucléotides : l'affichage de compatibilités (Sneath *et al.*, 1975), les tests de regroupement (Stephens, 1985), l'approche basée sur la randomisation (Sawyer, 1989) et l'extension de la méthode de reconstruction d'arbres par parcimonie qui permet la recombinaison (Hein, 1993).

En 1995, Rieseberg et Morefield (1995) ont conçu le programme *RETICLAD*. Ce programme permet d'identifier les hybrides en se basant sur l'idée qu'ils combinent les caractères de leurs parents. Hallett et Lagergren (2001) ont, quant à eux, exposé une méthode qui détecte le transfert horizontal de gènes en quantifiant la différence topologique entre un arbre d'espèces et un arbre de gène. Dans la même lancée, Legendre et Makarenkov (2002) et Makarenkov et Legendre (2004) ont suggéré l'utilisation de réticulogrammes pour représenter les réticulations dans des données évolutionnaires. Ces auteurs ont mis au point une méthode basée sur les distances qui reconstruit des phylogénies réticulées. Cette méthode se sert de la topologie d'un arbre phylogénétique comme une structure de base sur laquelle on ajoute progressivement des arêtes de réticulation pour construire un réticulogramme suivant un critère d'optimisation choisi. Bryant et Moulton (2004) ont développé une méthode, *NeighborNet*, permettant la reconstruction de réseaux phylogénétiques planaires. Par la suite,

Huson et Bryant (2006) ont développé des algorithmes basés sur des décompositions en coupe qui prennent en entrée différentes données sur un ensemble d'espèces et qui déterminent différentes décompositions de cet ensemble. Toutefois, ces algorithmes ne fournissent pas un réseau explicite et les résultats obtenus sont souvent difficiles à interpréter, même si Gambette et Huson (2008) ont grandement amélioré la visualisation de telles décompositions. Le logiciel *SplitsTree* (Huson et Bryant, 2006) est cependant l'un des outils les plus utilisés actuellement.

Il existe aussi d'autres techniques d'inférence de réseaux phylogénétiques telles que : les pyramides (Diday et Bertrand, 1986), les hiérarchies faibles (Bandelt et Dress, 1989), la parcimonie statistique (Templeton *et al.,* 1992), la parcimonie à variance moléculaire (Excoffier et Smouse, 1994), le *Netting* (Fitch, 1997), les réseaux medians-joints (Foulds *et al.,* 1979; Bandelt *et al.,* 1999) et les réseaux médians (Bandelt *et al.,* 1995 et 2000).

1.5 Transfert horizontal de gènes

Le problème de la détection et de la classification des transferts horizontaux de gènes (le pluriel est utilisé quand il s'agit de plusieurs gènes et le singulier quand il s'agit d'un seul gène) est parmi les plus ardus en biologie moléculaire. Le transfert horizontal de gènes (THG) se définit comme le passage d'une information génétique d'une espèce à une autre par des processus biologiques et l'insertion de cette information au sein du génome de l'hôte. Les transferts horizontaux sont très fréquents chez les bactéries. De ce fait, ces dernières deviennent, par exemple, de plus en plus résistantes aux antibiotiques en acquérant des gènes de résistance à partir d'autres espèces. Le THG est le mécanisme moteur et principal de l'évolution des micro-organismes (Doolittle *et al.*, 2003; Koonin, 2003). La détection de ces transferts est un défi important posé par l'évolution réticulée.

Le THG joue un rôle important dans l'évolution des espèces. Plusieurs études ont suggéré que les transferts horizontaux étaient extrêmement fréquents chez les procaryotes (Nelson *et al.*, 1999). Certaines de ces études sont même allées jusqu'à rejeter le concept d'arbre pour représenter l'évolution des procaryotes, considérant que seul un réseau pourrait offrir une représentation adéquate (Doolittle, 1999). Les bactéries et les archéobactéries ont développé des mécanismes sophistiqués pour acquérir rapidement de nouveaux gènes à l'aide

du transfert horizontal. Ces mécanismes ont été favorisés par la sélection naturelle par rapport à l'évolution génétique par mutation.

Il existe trois principaux mécanismes de transfert horizontal de gènes (voir la Figure 1.6) :

- La *transformation* est le plus simple des mécanismes de transfert. Dans le milieu extérieur d'un organisme se trouve de l'ADN libre qui résulte en général de la mort d'un autre organisme. Cet ADN peut être intégré à l'intérieur du génome de l'espèce hôte. Il y a donc un transfert horizontal entre les deux espèces concernées qui peuvent être tout à fait différentes.
- Le second mécanisme est celui de la *conjugaison* où les organismes s'échangent du matériel génétique en utilisant les plasmides. Ce mécanisme est particulièrement important à l'intérieur d'une même espèce (*i.e.*, souches différentes de la même espèce), mais le phénomène de conjugaison peut également avoir lieu entre des organismes appartenant à des espèces différentes.
- Le dernier mécanisme est celui de la *transduction*, où l'ADN est transféré d'une espèce à une autre via des virus ou des phages (*i.e.*, petits virus). Certains types de virus sont capables d'intégrer leur matériel génétique dans le génome de l'hôte, donnant ainsi naissance à de nouvelles particules virales. Ils peuvent amener par erreur une partie du matériel génétique de l'hôte, et comme ces organismes n'ont pas une spécificité d'hôte très importante, ils sont capables de passer d'une espèce à une autre en transférant le matériel génétique. Le virus ne peut pas, dans ce cas, infecter l'hôte, car il n'a plus tout son matériel génétique. Un tel transfert horizontal peut être bénéfique pour l'hôte.

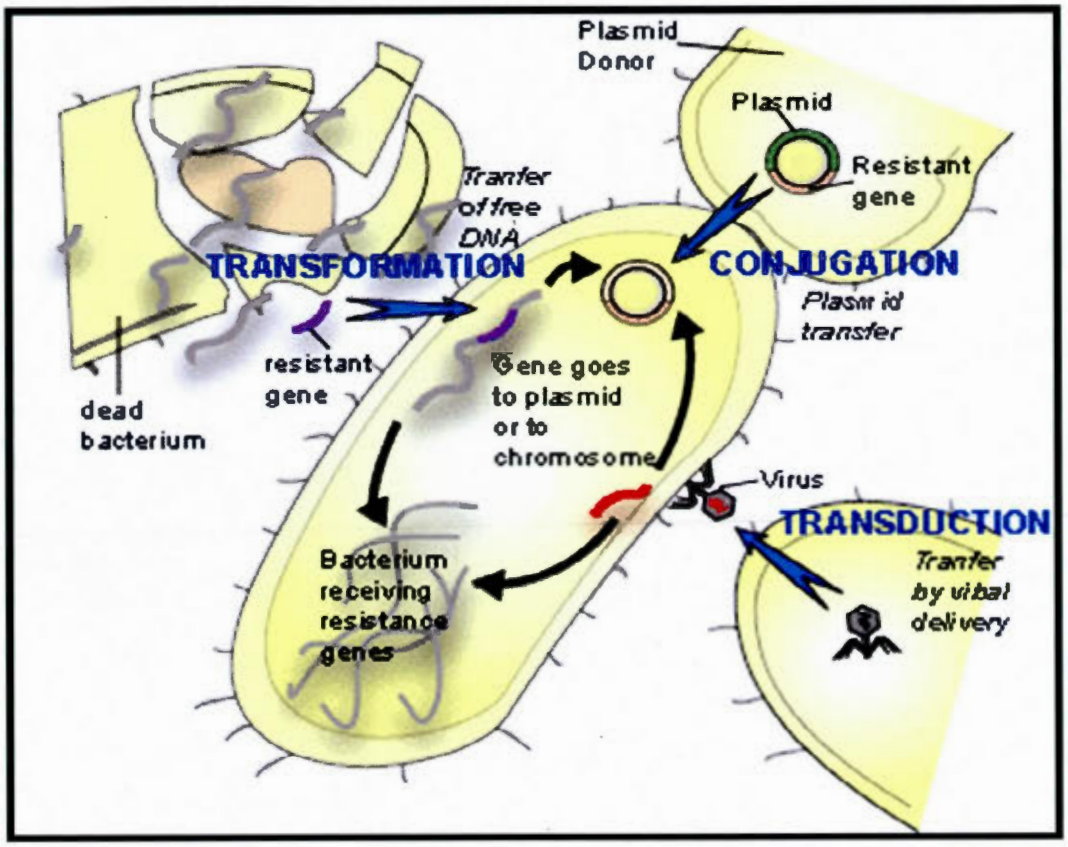


Figure 1.6 - Trois mécanismes de transfert horizontal de gènes (figure tirée du site : http://textbookofbacteriology.net/themicrobialworld/bactresanti.html).

Ces trois mécanismes sont très fréquents chez les procaryotes et génèrent souvent des échanges massifs de matériels génétiques (Jain *et al.*, 1999). Néanmoins, le matériel génétique n'est pas forcément conservé dans l'organisme hôte. Le gène transféré horizontalement doit être avantageux pour l'hôte. Dans la plupart des cas, suite à un transfert horizontal de matériel génétique d'une espèce à l'autre, l'ADN correspondant ne procure pas d'avantage sélectif et le nouveau gène est rejeté par l'hôte. Dans d'autres cas plutôt rares, il y a l'acquisition d'une nouvelle fonction grâce à un gène transféré horizontalement (Lawrence, 1999), par exemple une résistance aux antibiotiques. Dans ce cas, le gène transféré est conservé dans la population de l'hôte.

1.6 Méthodes de détection de transferts horizontaux de gènes

Il existe principalement deux manières d'aborder la question d'identification des gènes qui ont été transférés horizontalement. La première est basée sur une méthode d'analyse du génome de l'hôte qui peut révéler des séquences avec un taux de GC anormal (Lawrence et Ochman, 1997). En supposant que ces séquences ne résultent pas d'un processus de sélection naturel, on peut alors en déduire qu'elles sont la conséquence d'un transfert horizontal de gènes. La seconde est basée sur la comparaison d'un arbre phylogénétique d'espèces avec une phylogénie du gène observé, et inférée pour le même ensemble d'espèces. Cette comparaison peut révéler des conflits topologiques qui peuvent être expliqués par des transferts horizontaux de gènes.

A partir des années 1990, cette dernière approche, qui est surement la plus populaire, a permis le développement de plusieurs méthodes. Les premières méthodes, utilisant des modèles basés sur des réseaux, ont été introduites par Hein (1990, 1993), von Haeseler et Churchill (1993), Page (1994) et Charleston (1998). Mirkin *et al.* (1995) ont mis au point une méthode de réconciliation d'arbres qui combine différents arbres de gènes dans une seule phylogénie d'espèces. Maddison (1997) et, par la suite Page et Charleston (1997 et 1998), ont formulé un ensemble de règles d'évolution qui doivent être considérées quand on modélise les transferts horizontaux de gènes. Hallett et Lagergren (2001) ont proposé un modèle de détection de transferts permettant de transformer les phylogénies de gènes en phylogénies d'espèces.

Des méthodes plus récentes utilisent l'approximation de la distance SPR (Subtree Prune and Regraft), qui est liée de façon étroite à l'inférence des transferts horizontaux. La distance SPR représente le nombre de transferts dans le scénario le plus parcimonieux (Beiko et Hamilton, 2006). La distance SPR peut être définie comme le nombre minimal d'opérations de déplacement de sous-arbres nécessaires pour transformer un arbre phylogénétique en un autre arbre phylogénétique ayant le même ensemble de feuilles. Le calcul de la distance SPR entre deux arbres binaires enracinés est un problème NP-difficile (Bordewich et Semple, 2004).

1.6.1 L'algorithme *LatTrans*

L'algorithme *LatTrans* implémente un modèle de transfert horizontal de gènes qui compare l'évolution d'un ensemble d'arbres de gènes et celle d'un arbre d'espèces. Ce modèle a été développé par Hallett et Lagergren (2001) et Addario-Berry *et al.* (2003). Il permet de générer tous les scénarios avec une distance SPR minimale. Le nombre total de scénarios est exponentiel par rapport au nombre de transferts. *LatTrans* effectue la réconciliation des arbres de gènes et d'un arbre d'espèces donnés, tout en respectant un certain nombre de contraintes qui ont été introduites dans le modèle afin de rendre cette réconciliation biologiquement viable.

La Figure 1.7 représente le scénario de transferts horizontaux du gène *rbcL* inféré par *LatTrans*. On peut noter aussi que le modèle de Hallett et Lagergren (2001) inclut un paramètre d'activité α qui définit le nombre de gènes autorisés à être actif simultanément. Le programme *LatTrans* permet aussi la recherche des scénarios selon une combinaison de modèles de transferts horizontaux et de duplication de gènes.

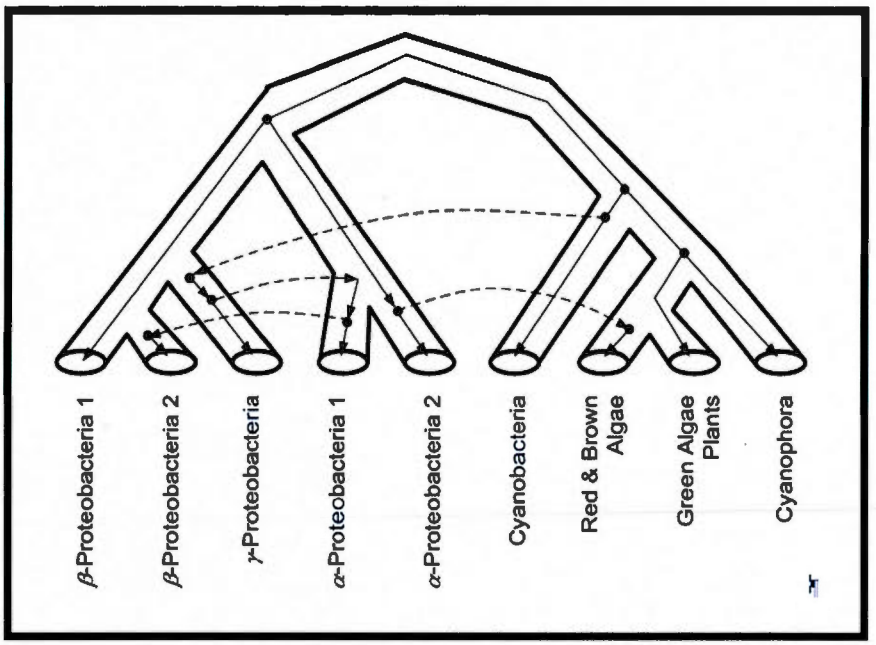


Figure 1.7 : Le scénario de transferts du gène *rbcL* identifié par Hallett et Lagergren (2001).

1.6.2 Premiers algorithmes pour la détection de transferts horizontaux de gènes développés au laboratoire bioinformatique de l'UQAM

Boc et Makarenkov (2003) ont présenté une approche, qui permet de détecter un scénario minimal de transferts horizontaux de gènes, basée sur la réconciliation d'un arbre d'espèces et d'un arbre de gène. Cette méthode a par la suite été améliorée dans Makarenkov *et al.* (2006). Les deux modèles d'évolution génétique proposés étaient : le modèle supposant le transfert d'une partie du gène et le modèle de transfert du gène au complet.

La Figure 1.8 illustre les deux modèles considérés. Dans le premier cas, celui du transfert partiel, une partie du gène transféré du donneur est récupérée par l'espèce hôte, ce qui entraine la transformation de l'arbre phylogénétique d'espèces en un réseau phylogénétique par l'ajout d'une arête orientée représentant le transfert du gène (voir la Figure 1.8, l'arbre du bas à gauche). Le second modèle d'évolution considéré suppose que le transfert du gène au complet entraine la transformation de l'arbre phylogénétique d'espèces en un autre arbre phylogénétique (Figure 1.8, l'arbre du bas à droite).

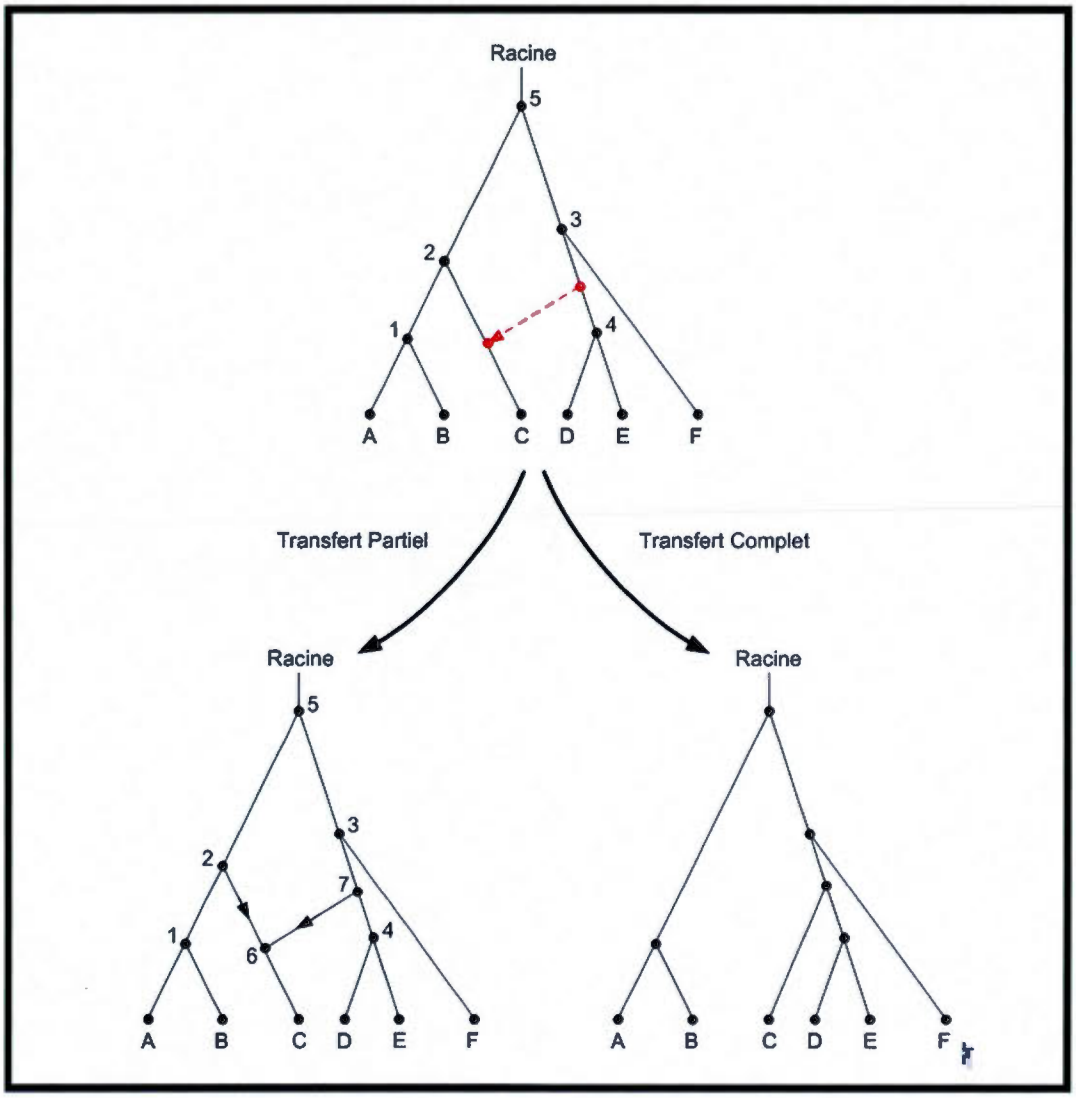


Figure 1.8 - Comparaison des deux modèles de transferts horizontaux selon Makarenkov *et al.* (2006).

1.6.3 Autres études et avancées dans le domaine de la détection de transferts horizontaux de gènes

Mirkin *et al.* (2003) ont développé un algorithme qui procède à la réconciliation de modèles phylétiques avec un arbre d'espèces. Ce modèle postule la perte, l'émergence et le transfert de gènes. Hallett *et al.* (2004) ont proposé un modèle combinatoire introduisant des THGs et

des duplications. MacLeod *et al.* (2005) ont développé l'algorithme *HorizStory* qui sert à approximer la distance SPR entre des arbres phylogénétiques enracinés. Cet algorithme débute par l'élimination des sous-arbres identiques dans les arbres de gène et d'espèces. Les opérations SPR sont ensuite exécutées de façon récursive jusqu'à la réconciliation complète des deux arbres. Nakhleh *et al.* (2005) ont conçu l'heuristique *RIATA-HGT* qui est basée sur l'approche *diviser pour régner*. D'autre part, Beiko et Hamilton (2006) ont développé l'algorithme Efficient Evaluation of Edit Paths (EEEP) qui minimise le nombre de transferts SPR entre deux arbres enracinés. L'approche adoptée par EEEP examine les bipartitions associées aux arêtes de l'arbre d'espèces et de l'arbre de gène. L'objectif principal des comparaisons topologiques dans cet algorithme est la subdivision de l'ensemble des bipartitions de l'arbre d'espèces en sous-ensembles qui sont concordantes et discordantes avec les bipartitions de l'arbre de gène. Than et Nakhleh (2008) ont démontré que la dernière version de *RIATA-HGT* est plus rapide que *LatTrans*. Cependant, les deux algorithmes sont identiques en termes de précision. Malgré la diversité des méthodes proposées, elles sont très sensibles aux types de données utilisées et aucunes d'elles n'apportent une solution optimale.

1.7 Algorithmes bioinformatiques et leur optimisation

La mise en œuvre d'architectures parallèles efficaces est très importante dans le domaine de la bioinformatique afin de réduire le temps d'exécution des algorithmes, qui sont souvent exigeants en termes de mémoire et de puissance de calcul. Parmi ces algorithmes, notons ceux destinés à la découverte de nouveaux gènes dans les séquences d'ADN, la prévision des structures et des fonctions des protéines, l'alignement des séquences, la reconstruction d'arbres et de réseaux phylogénétiques (Zomaya, 2006; Talbi et Zomaya, 2008).

Les programmes de génomique, de protéomique, d'analyse de l'expression de gènes manipulent d'énormes volumes de données hétérogènes créées par des technologies à haut débit. L'acquisition, le stockage, l'analyse et le traitement des données bioinformatiques brutes ou annotées, leur comparaison avec d'autres données de bases extérieures généralistes ou spécifiques, ainsi que leur consultation nécessitent l'utilisation de machines ayant une très grande puissance de calcul et une capacité de stockage de données très importante.

La plupart des programmes bioinformatiques sont caractérisés par des traitements de données indépendantes et volumineuses. Cela est dû au fait que les données génomiques et protéomiques ont connu, et connaissent toujours, une croissance fulgurante (Mardis, 2008; Trapnell et Salzberg, 2009) (voir le Figure 1.9). Cette croissance est presque exponentielle et le volume des données disponibles doublent chaque année (voir la Figure 1.9). Ceci résulte du fait que la vitesse du séquençage, c'est-à-dire la détermination des séquences des gènes, est de plus en plus rapide et implique un plus grand nombre de génomes (voir la Figure 1.10).

Organismes	**Nombre de génomes séquencés complètement**	**Nombre de génomes en cours de séquençage**
Virus et viroïdes	3889	N/D
Archées	113	91
Bactéries	1588	4914
Eucaryotes	36	1175

*Viroïdes : Petit ARN simple brin qui peut se répliquer de façon autonome, mais ne codant pas pour la protéine, ni encapsulé dans une couche.

*N/D : Non Disponible.

Tableau 1.1 - Projets de séquençage des génomes (tableau tiré de http://www.ncbi.nlm.nih.gov/genome; données de 2012)

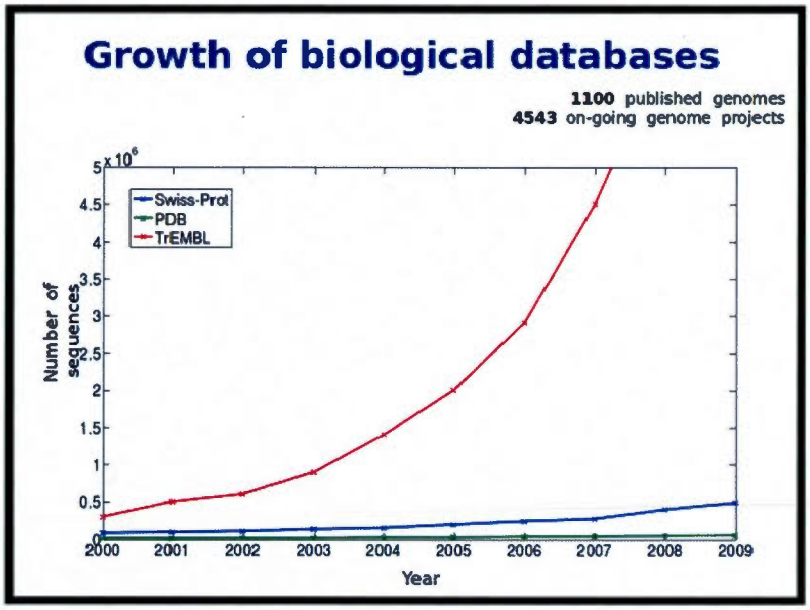


Figure 1.9 - Croissance des bases de données des séquences de nucléotides (figure tirée du site Byte Size Biology : http://bytesizebio.net/index.php/2011/07/02/cafa-update/).

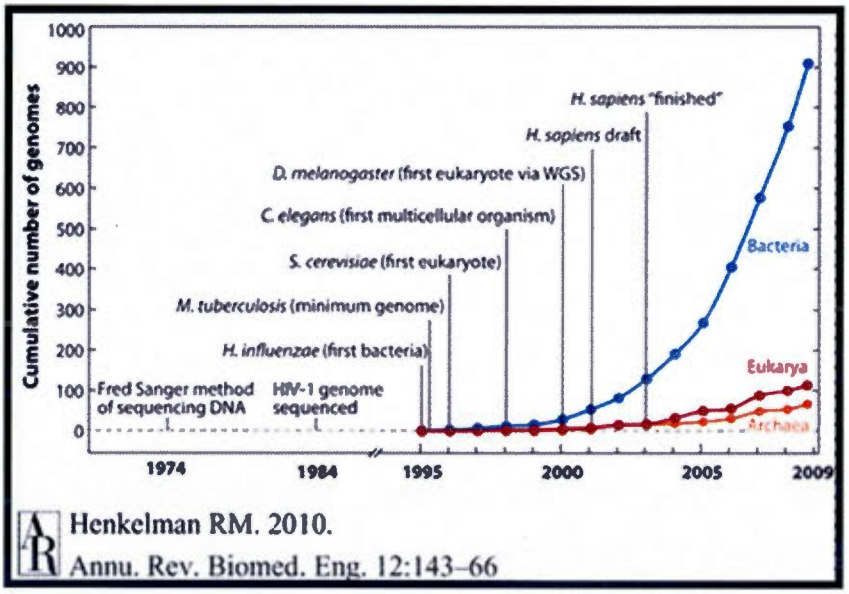


Figure 1.10 - Nombre de génomes d'espèces complètement séquencés (figure tirée d'Annual Review of Biomedical Engineering 12:143-66)

Comme les techniques de séquençage de génomes sont devenues beaucoup plus rapides et moins onéreuses, le domaine de la génomique progresse principalement dans deux directions :

- La première a pour but de déterminer de plus en plus de séquences génomiques humaines, en particulier celles qui peuvent se révéler utiles dans la recherche concernant l'anticipation et la prévention des maladies.
- La seconde a pour but de séquencer les génomes d'autres organismes vivants.

La programmation parallèle peut être une solution adéquate pour l'accélération des programmes bioinformatiques, dont les applications génomiques. Des architectures parallèles ont été récemment utilisées pour accélérer plusieurs algorithmes bioinformatiques (voir le Tableau 1.2). Par exemple, des programmes bioinformatiques tels que *BLAST* (Altschul *et al.*, 1990) et *ClustalW* (Thompson *et al.*, 1994) ont été parallélisés à l'aide de la bibliothèque d'échange de messages MPI : *MPI-Blast* (Darling *et al.*, 2003; Lin *et al.*, 2005; Lin *et al.*, 2010) et *ClustalW-MPI* (Li, 2003).

Les raisons principales pour le développement et l'utilisation des environnements parallèles en bioinformatique sont les suivantes :

- Réduction du temps d'exécution / Analyse de grandes banques de données bioinformatiques : l'allocation de plus de ressources à une tâche permet de réduire son temps d'exécution.
- Résoudre des problèmes de grande taille: sachant que les données génomiques et protéomiques connaissent toujours une croissance fulgurante, l'utilisation d'environnements parallèles permettrait leur traitement en temps raisonnable. La plupart de ces problèmes sont si complexes qu'il devient presqu'impossible de les résoudre sur un ordinateur de bureau.
- Prendre avantage de la concurrence: une seule ressource de calcul ne peut exécuter qu'une seule tâche à la fois, alors que les ressources informatiques multiples (ou parallèles) peuvent exécuter plusieurs tâches simultanément.

Au niveau des systèmes parallèles multiprocesseurs, l'optimisation du compilateur et la parallélisation sont essentielles afin de profiter de la puissance de traitement nécessaire pour exécuter des applications contenant des calculs intensifs. Notons que la transformation d'un programme séquentiel en un programme parallèle nécessite un certain nombre de modifications du code source. Nous avons développé une nouvelle méthode et une interface Web associée pour faciliter la tâche du programmeur lors de la transformation d'un programme séquentiel en un programme parallèle en identifiant les principales étapes de cette transformation et en utilisant la bibliothèque d'échange de messages MPI. À chaque étape de cette transformation, nous proposons une solution permettant de simplifier le travail du programmeur, de diminuer le temps nécessaire au développement du code parallèle et d'augmenter la qualité du programme parallèle.

Le Tableau 1.2 présente une liste de programmes bioinformatiques parallélisés à l'aide d'API tels que MPI, CUDA et OpenMP.

Programme	Objectif	API	Référence	Site Web
Argonne Smith-Waterman	AMS	MPI/CUDA (en développement)	N/D	N/D
Autodock4	Recherche moléculaire	MPI	Collignon *et al.* (2011)	http://autodock.scripps.edu/wiki/AutoDock4
BEAST	Inférence phylogénétique	MPI	Drummond et Rambaut (2007)	http://beast.bio.ed.ac.uk
Clustal-MPI	AMS	MPI	Li (2003)	http://www.bii.a-star.edu.sg/achievements/applications/clustalw
CUDA Smith-Waterman	AMS	CUDA	Manavski et Valle (2008)	N/D
CUDA-BLASTP and mpiCUDA-BLASTP	Recherche de séquences	CUDA et MPI/CUDA (hybride)	Liu *et al.* (2011b)	https://sites.google.com/site/liuweiguohome/software
CUDA-MEME	Recherche de motifs	CUDA	Lui *et al.* (2010a)	https://sites.google.com/site/yongchaosoftware/Home/cuda-meme
CUDASW++	AMS	CUDA	Liu *et al.* (2009 et 2010b)	https://sites.google.com/site/yongchaosoftware/cudasw
Dialign-P	AMS	MPI	Schmollinger *et al.* (2004)	bibiserv.techfak.uni-bielefeld.de/dialign
Dialign-TX	AMS	MPI/OpenMP (hybride)	de Araujo Macedo *et al.* (2011)	http://bibiserv.techfak.uni-bielefeld.de/dialign
fastDNAml	Inférence phylogénétique	MPI, PVM	Stewart *et al.* (2001)	http://iubio.bio.indiana.edu/soft/molbio/evolve/fastdnaml/fastDNAml.html
GARD	Recherche de recombinants	MPI	Pond *et al.* (2006)	www.hyphy.org
Garli	Inférence phylogénétique	MPI	Zwickl (2006)	https://www.nescent.org/wg_garli/MPI_version
GPU-ClustalW	AMS	CUDA	Liu *et al.* (2007)	N/D

GPU-HMMER	Recherche de séquences	CUDA	Walters *et al.* (2009)	http://www.mpihmmer.org/index.htm
Hybrid ClustalW	AMS	MPI/OpenMP (hybride)	Tan *et al.* (2005)	N/D
HyPhy	Analyse génétique de séquences	OpenMP, MPI et gpGPU (OpenCL, prochaine version)	Pond *et al.* (2005)	http://www.hyphy.org
MAFFT	Inférence phylogénétique	Pthreads	Katoh et Toh (2010)	http://mafft.cbrc.jp/alignment/software/
mCUDA-MEME	Recherche de motifs	MPI/CUDA/Open MP (hybride)	Liu *et al.* (2011a)	https://sites.google.com/site/yongchaosoftware/mcuda-meme
mpiBLAST	Recherche de séquences	MPI	Darling *et al.* (2003) Lin *et al.* (2010)	http://www.mpiblast.org
mpiFASTA	AMS	MPI	N/D	http://www.hpc.fsu.edu/index.php?option=com_content&view=article&id=136
MPI-HMMER	Recherche de séquences	MPI	Walters *et al.* (2007)	http://www.mpihmmer.org/index.htm
MrBayes	Inférence phylogénétique	MPI	Altekar *et al.* (2004)	http://mrbayes.sourceforge.net
MSAProbs	AMS	CUDA	Liu *et al.* (2010c)	https://sites.google.com/site/yongchaosoftware/msaprobs
MUMmerGPU	Alignement de génomes	CUDA	Trapnell et Schatz (2009)	http://sourceforge.net/apps/mediawiki/mummergpu/index.php?title=MUMmerGPU
PAML	Analyse génétique de séquences	MPI	Chen *et al.* (2009)	http://palm.iis.sinica.edu.tw
pCLUSTAL	AMS	MPI	Cheetham *et al.* (2003)	N/D
PHYLIP	Inférence phylogénétique	MPI	Ropelewski *et al.* (2010)	http://www.nrbsc.org/downloads
PhyML	Inférence phylogénétique	MPI/OpenMP (hybride)	Guindon *et al.* (2010)	http://www.atgc-montpellier.fr/phyml
PyEvolve	Modélisation statistique de l'évolution moléculaire	MPI	Butterfield *et al.* (2004)	http://pyevolve.sourceforge.net

RaxML	Inférence phylogénétique	MPI, Pthreads, CUDA, OpenMP	Stamatakis (2006) Pfeiffer et Stamatakis (2010)	http://sco.h-its.org/exelixis/software.html
Recodon	Simulation de coalescence	MPI	Arenas et Posada (2007)	http://darwin.uvigo.es/software/recodon.html
RNAFold	Repliement des ARN	MPI	N/D	http://www.tbi.univie.ac.at/RNA
RNAFold	Repliement des ARN	CUDA	N/D	http://bibiserv.techfak.uni-bielefeld.de/ adp/cuda.html
Tree-Puzzle	Inférence phylogénétique	MPI	Schmidt *et al.* (2002)	http://www.tree-puzzle.de
Trex HGT	Transfert Horizontal de gènes	MPI	N/D	N/D

*AMS : Alignement Multiple de Séquences

*N/D : Non Disponible

Tableau 1.2 - Liste de programmes bioinformatiques parallélisés en utilisant MPI, OpenMP et CUDA.

1.8 Résumé

Dans ce chapitre, nous avons présenté les notions de base de la phylogenèse dont nous aurons besoin, ainsi que les principales méthodes de reconstruction d'arbres phylogénétiques et de réseaux réticulés, en particulier celles servant à la détection de transferts horizontaux de gènes. Nous avons aussi constaté que la plupart des problèmes bioinformatiques actuels sont reliés à l'analyse des données de grande taille, ce qui pourrait nécessiter l'utilisation de la programmation parallèle pour leur traitement rapide et efficace.

CHAPITRE II

ALGORITHMES POUR LA DÉTECTION DE TRANSFERTS HORIZONTAUX DE GÈNES COMPLETS

Dans ce chapitre, nous décrivons une méthode de détection de transferts horizontaux de gènes complets basée sur l'algorithme que nous avons proposé dans Makarenkov *et al.* (2007). Nous présentons aussi une version plus complète de cette méthode décrite par Boc *et al.* (2010). Par la suite, nous introduisons deux nouveaux algorithmes pour la construction d'un scénario de consensus de THGs et pour une validation interactive des transferts détectés.

2.1 Introduction

Le processus de détection de transferts horizontaux de gènes que nous proposons procède par une réconciliation graduelle d'une phylogénie enracinée d'espèces et d'une phylogénie enracinée d'un gène, notées respectivement T et T' (voir la Figure 2.1). À chaque étape du processus de réconciliation ou de détection, plusieurs paires d'arêtes de T sont évaluées selon l'hypothèse qu'un THG se soit produit entre elles. Les transferts horizontaux sont ajoutés progressivement à l'arbre phylogénétique d'espèces T. La distance de Robinson et Foulds (Robinson et Foulds, 1981) est utilisée comme critère d'optimisation topologique et les moindres carrés (Gauss, 1811) comme critère d'optimisation métrique. Deux autres critères topologiques ont été introduits par la suite, à savoir la distance de quartets (Bryant *et al.*, 2000) et la dissimilarité de bipartitions que nous avons introduite dans Makarenkov *et al.* (2007).

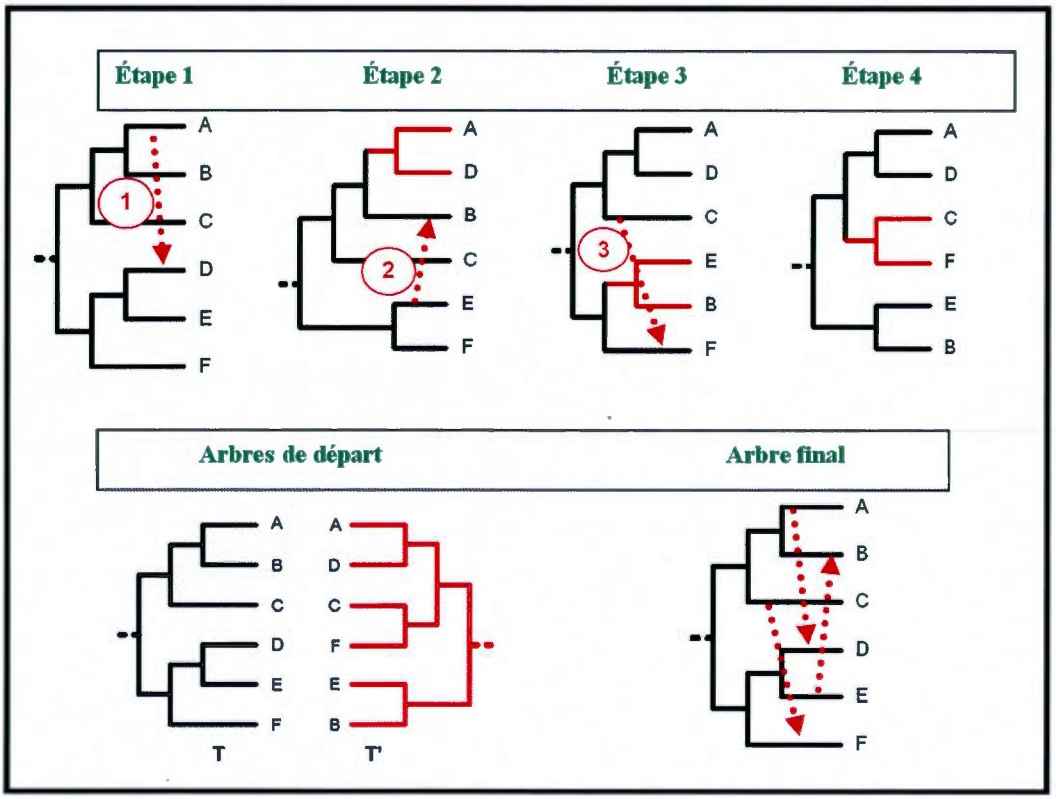


Figure 2.1 - Fonctionnement de l'algorithme de détection de THGs procédant à la réconciliation des arbres d'espèces T et de gène T'.

L'arbre phylogénétique d'espèces original T est graduellement transformé en l'arbre phylogénétique du gène T' par une série d'opérations ou de mouvements SPR (*i.e.*, THGs; voir le chapitre I, section 1.6). Le but ces opérations est de trouver la plus courte séquence possible d'arbres T, T_1, T_2, ..., T' qui transforme T en T' (voir la Figure 2.1). Les transferts horizontaux sont ajoutés au fur et à mesure dans l'arbre phylogénétique d'espèces T en le transformant ainsi en arbre phylogénétique du gène T'.

Quatre critères d'optimisation peuvent être utilisés lors du processus de détection de transferts horizontaux :

Le premier est le critère des moindres carrés (Gauss, 1811) défini par la fonction *MC* suivante :

$$MC = \sum_i \sum_j (d(i,j) - \delta(i,j))^2 , \qquad (1)$$

où $d(i,j)$ est la distance entre les espèces i et j dans l'arbre phylogénétique d'espèces T et $\delta(i,j)$ est la distance entre les espèces i et j dans l'arbre phylogénétique de gène T' (construit pour le même ensemble d'espèces).

Le deuxième critère est la distance topologique de Robinson et Foulds (1981). La distance de Robinson et Foulds (RF) indique le nombre minimum d'opérations élémentaires de contraction et d'expansion de nœuds nécessaires pour la transformation d'un arbre phylogénétique en un autre. Les deux opérations nécessaires pour transformer un arbre T en un arbre T_1 sont illustrées dans la Figure 2.2.

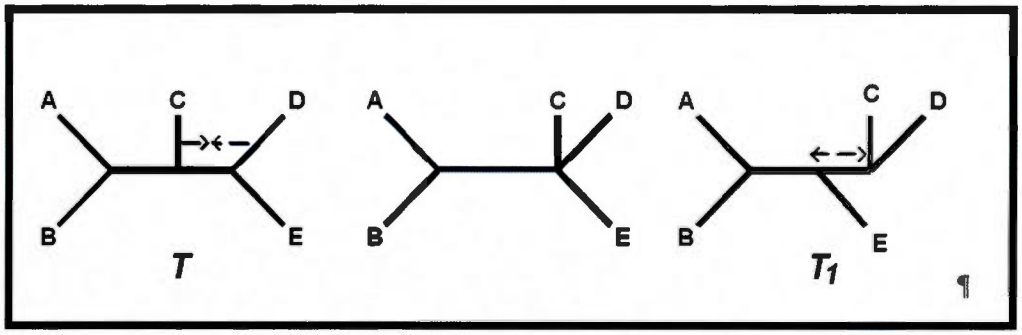


Figure 2.2 - La distance de Robinson et Foulds entre les arbres T et T_1 est égale à deux.

Quand la distance RF est considérée, nous pouvons l'utiliser comme un critère d'optimisation de la manière suivante : toutes les transformations possibles de l'arbre d'espèces, consistant au transfert d'un de ses sous-arbres d'une arête vers une autre, sont estimées en calculant la distance RF entre l'arbre d'espèces transformé T_1 et l'arbre de gène T'. Le transfert de sous-arbres minimisant la distance RF est choisi. Cependant, rappelons que Hein *et al.* (1996) ont démontré que le problème de la recherche du nombre minimal de transferts de sous-arbres nécessaires pour transformer un arbre en un autre, aussi connu sous le nom du problème de transfert des sous-arbres (*i.e.*, distance SPR), est NP-complet.

Le troisième critère considéré est la distance de quartets (QD) qui représente le nombre des quartets (sous-arbres induits par quatre feuilles) différents entre les arbres comparés (Bryant *et al.*, 2000).

Le dernier critère que nous définissons ici est la dissimilarité de bipartitions. Cette mesure de proximité topologique entre deux arbres phylogénétiques est un raffinement de la distance RF (Robinson et Foulds, 1981) qui prend en compte seulement les bipartitions identiques entre les deux phylogénies comparées. Supposons que les arbres T et T' sont respectivement les deux phylogénies binaires d'espèces et de gène, ayant le même ensemble de feuilles. Un vecteur de bipartitions d'un arbre T est un vecteur binaire induit par une arête de T. Soit **BT** la table de bipartitions des arêtes internes de l'arbre T (c'est-à-dire, la table qui contient tous les vecteurs de bipartitions induits par les arêtes internes de T) et **BT'** la table de bipartitions des arêtes internes de l'arbre T'. La dissimilarité de bipartitions (*DB*) entre les arbres T et T' est calculée de la façon suivante :

$$DB = \left(\sum_{a \in BT} Min_{b \in BT'}(Min(d(a,b); d(a,\bar{b}))) + \sum_{b \in BT'} Min_{a \in BT}(Min(d(b,a); d(b,\bar{a}))) \right) / 2; \qquad (2)$$

où $d(a,b)$ est la distance de Hamming (1950) entre les vecteurs de bipartitions a et b, et $\bar{a}$ et $\bar{b}$ sont respectivement les compléments (vecteurs complémentaires) de a et b.

Par exemple, la dissimilarité de bipartitions entre les arbres T et T' à 6 feuilles montrés sur la Figure 2.3 est calculée de la façon suivante :

$$DB(T,T') = ((2 + 1 + 2) + (2 + 1 + 1))/2 = 4,5;$$

La distance de Hamming minimale entre les bipartitions (directe et complémentaire) correspondantes à l'arête a et tous les vecteurs de la table de bipartitions **BT'** est 2. C'est la distance entre les vecteurs a et $\bar{f}$, et a et d (seule l'association entre a et $\bar{f}$ est représentée par une flèche dans la Figure 2.3). Pour la bipartition b, la distance est 1 (la distance entre b et d) et pour la bipartition c, la distance est 2 (la distance entre c et d). De la même façon, la distance de Hamming minimale entre la bipartition e et toutes les bipartitions de **BT** est 2

(voir la bipartition associée à $\bar{b}$), pour la bipartition *f* cette distance est 1 (voir la bipartition associée à $\bar{b}$) et pour la bipartition *d*, elle est égale à 1 (voir la bipartition associée à *b*).

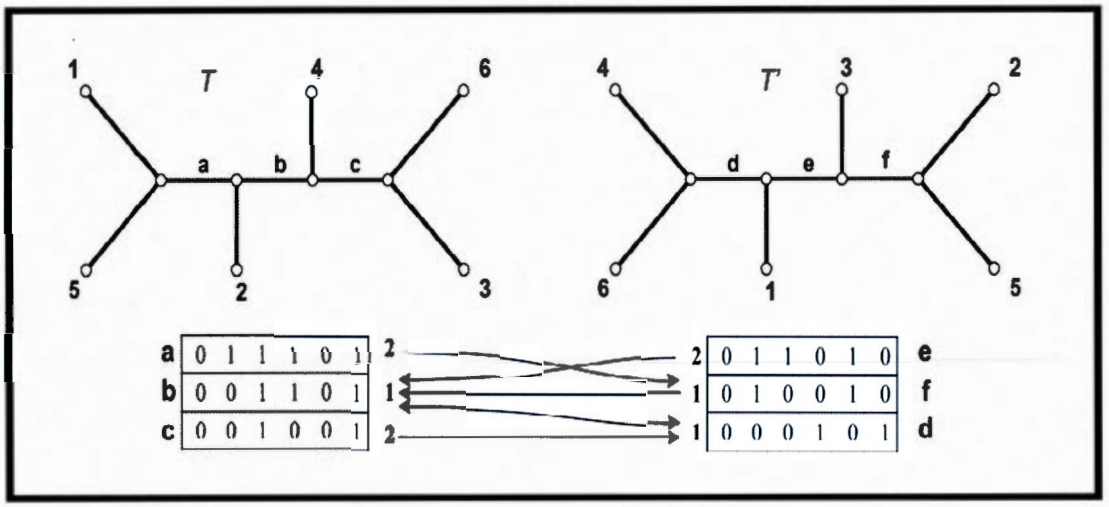


Figure 2.3 - Les arbres *T* et *T'* et leur table de bipartitions. Chaque ligne de la table de bipartitions correspond à une branche interne de l'arbre. Les flèches indiquent les associations entre les vecteurs de bipartitions dans les deux tables (BT et BT'). La valeur en gras, proche de chaque vecteur, représente la distance associée (figure tirée de Makarenkov *et al.*, 2007).

Cet exemple montre aussi que plusieurs vecteurs de bipartitions de la première table de bipartitions peuvent être associés au même vecteur de bipartitions de la seconde table. Par exemple, *d*, *e* et *f* sont associés à *b* (ou $\bar{b}$), et *b* et *c* sont associés à *d* (voir la Figure 2.3). La dissimilarité de bipartitions n'est pas toujours une distance (*i.e.*, une métrique). Pour les arbres avec cinq feuilles et plus, il est possible d'exhiber trois topologies d'arbres pour lesquelles l'inégalité triangulaire n'est pas respectée. La condition suivante est suffisante pour assurer qu'une dissimilarité de bipartition est une métrique (le symbole → désigne l'opération d'association) :

Proposition 1. *Soient T_1, T_2 et T_3 des arbres phylogénétiques ayant le même nombre d'arêtes internes et le même ensemble de feuilles. Alors, $DB(T_1, T_2) \leq DB(T_1, T_3) + DB(T_2, T_3)$ si les conditions suivantes sont satisfaites* :

1. *Pour chaque paire de bipartitions a et b de deux arbres différents: $a \rightarrow b$ implique que $b \rightarrow a$.*

2. *Pour chaque triplet de bipartitions $a \in T_1$, $b \in T_2$, $c \in T_3$: $a \rightarrow b$ et $b \rightarrow c$ implique que $a \rightarrow c$.*

Proposition 2. *La valeur d'une dissimilarité de bipartition entre deux arbres phylogénétiques ayant le même ensemble de n feuilles se trouve dans l'intervalle entre* 0 *et* $n(n-3)/2$, *si n est paire, et entre* 0 *et* $(n-1)(n-3)/2$, *si n est impaire.*

Ces propositions que nous avons introduites dans Makarenkov *et al.* (2007) ont par la suite été démontrées dans Boc *et al.* (2010). Lors de la détection de THGs, un certain nombre de contraintes d'évolution doit être pris en compte car un THG exige que les espèces source et destination soient contemporaines. Par exemple, le transfert sur une même lignée (voir la Figure 2.4a) et les transferts qui se croisent tel qu'illustré à la Figure 2.4b-d, conduisent à des scénarios erronés et ne doivent pas être permis (voir aussi Maddison, 1997 ou Hallett et Lagergren, 2001).

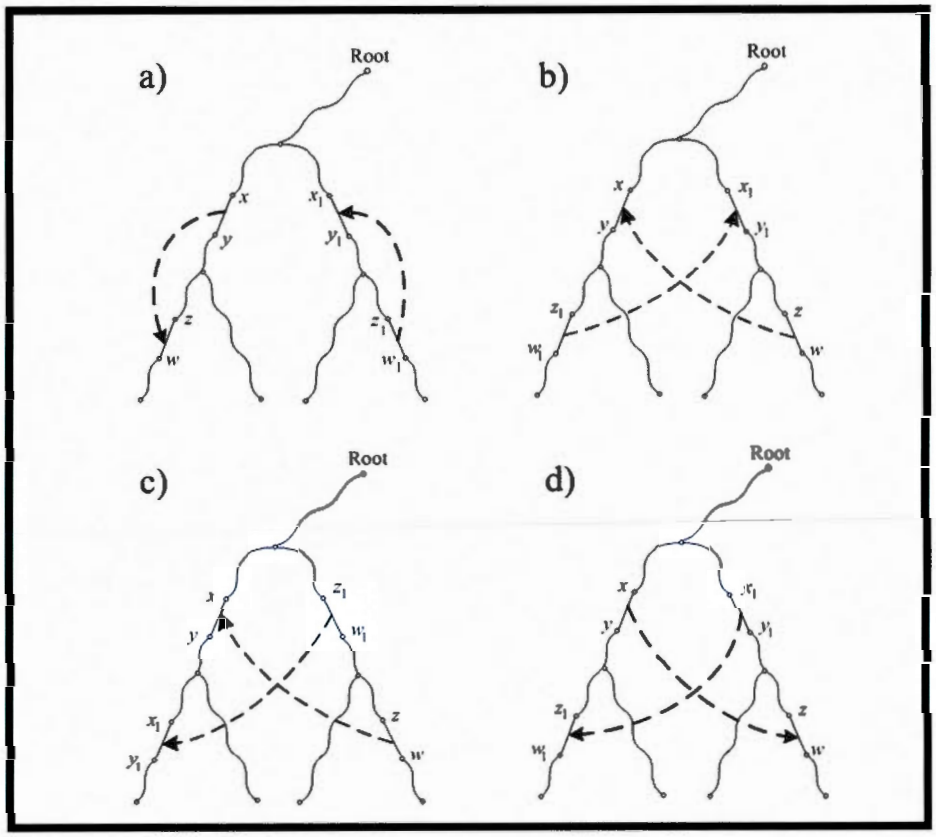


Figure 2.4 - Cas de figures où un transfert de gène est interdit. Cas a : les transferts entre les branches situées sur la même lignée doivent être interdits. Cas b, c et d : les transferts croisés de cette façon doivent aussi être interdits. Une branche est représentée par une ligne droite et un chemin par une ligne ondulée (figure tirée de Boc *et al.*, 2010).

Une autre contrainte, appelée *contrainte de sous-arbres* (Makarenkov *et al.*, 2006), permet tout d'abord d'arranger les conflits topologiques entre T et T', qui sont dus aux transferts entre les ancêtres des espèces contemporaines et qui sont donc plus faciles à détecter. La *contrainte de sous-arbres* permet ensuite d'identifier des transferts apparus plus profondément dans la phylogénie. Considérons un THG dans l'arbre d'espèces T, allant de a à b et le transformant en un arbre T' (voir la Figure 2.5). La *contrainte de sous-arbres* est énoncée comme suit : pour permettre le THG entre les arêtes (x,y) et (z,w) de l'arbre d'espèces T, le clade consistant en le sous-arbre enraciné par l'arête (x,a), et incluant les nœuds y et w, doit être présent dans l'arbre de gène T' (Makarenkov *et al.*, 2006).

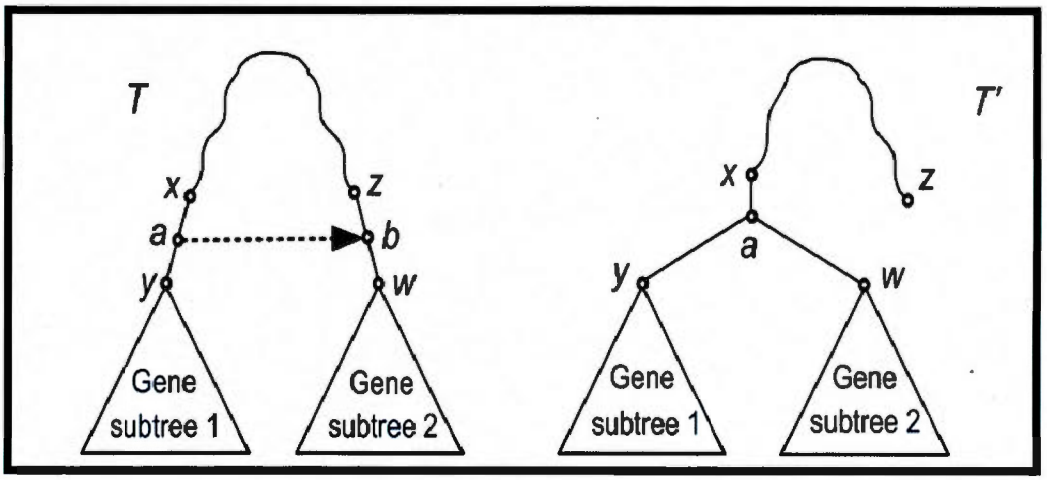


Figure 2.5 - Illustration de la contrainte de sous-arbres. Le transfert entre les arêtes (x,y) et (z,w) dans l'arbre d'espèces T est permis si et seulement si les sous-arbres enracinés par les arêtes (a,y) et (b,w), ainsi que leur regroupement enraciné par (x,a), sont présents dans l'arbre de gène T' (figure tirée de Makarenkov *et al.*, 2006).

L'utilisation de la contrainte de sous-arbres nous permet de prendre en compte automatiquement toutes les contraintes d'évolution requises (voir la Figure 2.4) car les deux sous-arbres impliqués dans un THG doivent être présents dans l'arbre de gène T', ainsi que le nouveau sous-arbre qu'ils forment après le transfert (voir la Figure 2.5). Si les THGs sur une même lignée (Figure 2.4a) où les transferts croisés présentés sur la Figure 2.4(b-d) étaient autorisés, alors la contrainte de sous-arbres ne serait pas respectée.

2.2 Algorithme pour la détection de transferts horizontaux de gènes complets

2.2.1 Description de l'algorithme

À chaque itération du processus de détection de THGs, le transfert diminuant le plus la valeur du critère d'optimisation sélectionné entre l'arbre d'espèces et l'arbre de gène est considéré comme le plus approprié. Dans ce cas, l'arête du transfert est ajoutée à l'arbre phylogénétique d'espèces avant l'itération suivante. En effectuant le transfert complet du gène, l'arête reliant l'espèce affectée par ce transfert et son ancêtre direct est supprimée de l'arbre car l'ajout

d'une arête de transfert est suivi par la suppression d'une autre arête. Dans ce cas, le processus de détection s'effectue toujours avec un graphe connexe et sans cycles, c'est-à-dire un arbre.

Les principales étapes de notre algorithme sont les suivantes :

Étape préliminaire. Inférer les arbres phylogénétiques d'espèces et de gène, notés respectivement T et T'. Les feuilles de T et T' sont étiquetées par le même ensemble de n éléments (*i.e.*, d'espèces). Les deux arbres doivent être enracinés en fonction des évidences biologiques existantes. Si aucune connaissance permettant d'enraciner les arbres n'est disponible, des stratégies d'enracinement par *outgroup* (en utilisant un groupe des espèces extérieur au groupe étudié) ou par *midpoint* (en utilisant l'arête médiane de l'arbre) peuvent être considérées. S'il existe dans T et T' des sous-arbres identiques ayant au moins 2 feuilles, réduire la taille du problème en remplaçant dans T et T' les sous-arbres identiques par les mêmes éléments auxiliaires.

Étape 1..k. Tester tous les THGs possibles entre les paires d'arêtes dans l'arbre T_{k-1} ($T_{k-1} = T$ à l'étape 1) à l'exception des transferts entre les arêtes adjacentes et ceux qui violent les contraintes d'évolution (voir la Figure 2.4). La contrainte de sous-arbre (voir la Figure 2.5) a été utilisée pour prendre en compte les contraintes d'évolution. Choisir en tant que THG optimal, le déplacement d'un sous-arbre dans T_{k-1} qui minimise la valeur du critère d'optimisation sélectionné entre l'arbre de gène et l'arbre obtenu après le déplacement de ce sous-arbre et son greffage sur une nouvelle arête, *i.e.* entre l'arbre T_k, et l'arbre de gène T'. Les critères d'optimisation suivants ont été testés dans nos simulations (voir la section 2.2.2) : (1) les moindres carrés (MC), (2) la distance topologique de Robinson et Foulds (RF) et (3) la dissimilarité de bipartition (DB). Réduire ensuite la taille du problème en remplaçant des sous-arbres identiques ayant au moins 2 feuilles dans l'arbre d'espèces transformé T_k et l'arbre de gène T'. Dans la liste des THGs retrouvés rechercher et éliminer les THGs inutiles en utilisant une procédure de programmation dynamique de parcours en arrière. Un transfert inutile est celui dont l'élimination ne change pas la topologie de l'arbre T_k.

L'algorithme s'arrête quand le coefficient RF, MC ou BD devient égale à 0 ou quand aucun autre déplacement de sous-arbres n'est possible suite à des contraintes biologiques.

Théoriquement, une telle procédure requière $O(kn^4)$ d'opérations pour détecter k transferts horizontaux dans un arbre phylogénétique à n feuilles. Notre algorithme de détection de THGs complets se présente comme suit (Figure 2.6) :

DÉBUT

Inférer les arbres d'espèces T et de gène T' sur le même ensemble d'espèces (i.e., feuilles);

Enraciner les arbres T et T' selon les évidences biologiques disponibles ou en utilisant un *outgroup* ou un *point médian*;

Si (il existe des sous-arbres identiques, ayant au moins deux feuilles, entre T et T') alors

    Diminuer la taille du problème en contractant les sous-arbres identiques dans T et T';

Sélectionner le critère d'optimisation CO entre : MC (moindres carrés), RF (distance de Robinson et Foulds) ou DB (dissimilarité de bipartitions);

Calculer la valeur initiale du critère d'optimisation CO entre T et T';

$T_0 = T$;

$k = 0$; //k est l'indice de l'étape

Tant que $(CO \neq 0)$ {

    Trouver le THG candidat (i.e., obtenu par une opération SPR) dans l'arbre T_k et représenté par THG_k : le THG_k est un transfert satisfaisant la contrainte de sous-arbres et minimisant le critère d'optimisation sélectionné CO;

    Recalculer la valeur du critère d'optimisation CO après l'ajout du transfert THG_k;

    $k = k + 1$;

    Diminuer la taille du problème en contractant les sous-arbres identiques dans T_k et T';

}

Éliminer les transferts inutiles obtenus;

FIN

Figure 2.6 - Algorithme de détection de transferts horizontaux de gènes complets.

2.2.2 Simulations

Cette section présente les performances des trois stratégies d'optimisation décrites ci-dessus qui peuvent être utilisées pour prédire les transferts horizontaux. Nous montrons ici seulement une partie des résultats obtenus dans nos simulations Monte Carlo (méthode visant à calculer une valeur numérique en utilisant des procédés aléatoires, c'est-à-dire des techniques probabilistes). Nous avons généré 100 topologies aléatoires différentes des arbres

d'espèces ayant 10, 20, ... et 100 feuilles, respectivement. Chaque topologie a été obtenue en utilisant la procédure de génération d'arbres proposée par Kuhner et Felsenstein (1994). Cette procédure génère un arbre phylogénétique aléatoire d'une taille donnée n. La procédure de génération de l'arbre débute par une branche unique, puis de nouvelles arêtes y sont ajoutées graduellement. À l'étape k, nous disposons d'un arbre avec $2k$-3 arêtes et k feuilles et nous y ajoutons la nouvelle arête (avec la feuille k+1) en choisissant l'arête de destination (de greffage) aléatoirement parmi les $2k$-3 arêtes de l'arbre en construction. Après n-1 étapes, nous obtenons un arbre phylogénétique binaire à n feuilles et $2n$-3 arêtes.

Pour chaque arbre d'espèces, nous avons généré des arbres de gène correspondant aux différents nombres de transferts tout en respectant les contraintes d'évolution. Le nombre exact de transferts variait de 1 à 10 pour chaque arbre de gène. Nous avons testé les trois stratégies d'optimisation, MC, RF et DB, présentées dans la section précédente pour mesurer le taux de détection de transferts générés (*i.e.*, *Detection rate* sur la Figure 2.7a) et le pourcentage des cas quand le nombre exact des transferts générés a été retrouvé (*i.e.*, *Same number of HGTs* sur la Figure 2.7b). Remarquons qu'un transfert correct est celui dont l'emplacement et la direction exacts ont été retrouvés. Pour les deux critères considérés (Figure 2.7a et b), la stratégie algorithmique basée sur la dissimilarité de bipartition (DB) était plus performante que les stratégies basées sur les moindres carrés et la métrique de Robinson et Foulds (RF). Les deux dernières stratégies avaient les tendances similaires, mais celle basée sur la distance RF a toujours fourni des meilleurs résultats que celle basée sur les moindres carrés. Notons que la solution optimale (i.e., le scénario de transferts complets de coût minimum) n'est pas toujours unique. Ceci explique le fait que le taux de détection était plus bas que le pourcentage de cas quand le nombre exact des transferts générés a été retrouvé. La détérioration des résultats, qui se manifeste quand le nombre de transferts augmente (cette détérioration est aussi caractéristique pour la version exhaustive de l'algorithme implémentée dans le logiciel *LatTrans*, Hallett et Lagergren 2001), est due au fait que le scénario de cout minimum ne correspond toujours pas au scénario généré dans le cas de petits arbres (e.g., 10, 20 et 30 espèces) et un grand nombre de transferts (e.g., 8, 9 et 10 THGs).

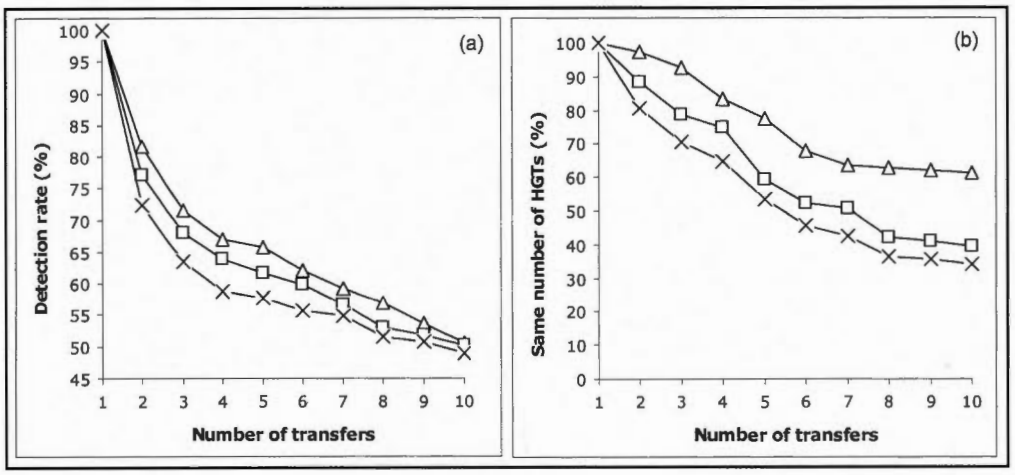


Figure 2.7 - Le pourcentage des cas quand l'algorithme prédit : (a) les transferts corrects ; (b) le nombre exact des transferts - versus le nombre de transferts. Pour chaque point du graphique la moyenne des résultats obtenus pour 100 arbres à 10, 20, ... et 100 feuilles est montrée. Les trois stratégies algorithmiques comparées sont : la dissimilarité de bipartition (Δ), la distance de Robinson et Foulds (□) et les moindres carrés (×).

2.3 Algorithme pour la détection de transferts horizontaux de gènes complets selon Boc *et al.* (2010)

L'algorithme de détection de THGs complets décrit dans Boc *et al.* (2010) représente un raffinement de notre algorithme présenté dans la section précédente. Les principales améliorations consistent en l'ajout des Théorèmes 1 et 2 (présentés ci-dessous) et en la possibilité d'inférer plusieurs transferts lors d'une même itération.

Les deux théorèmes ci-après définissent des propriétés des bipartitions dans le contexte des THGs qui satisfont la *contrainte de sous-arbres*. Ces propriétés sont utilisées et vérifiées par l'algorithme de détection de transferts horizontaux (*HGT-Detection*) de Boc *et al.* (2010).

Théorème 1. *Si le sous-arbre Sub_{yw} nouvellement formé et résultant du THG (à savoir le sous-arbre enraciné par l'arête (x,a) dans la figure 2.5) est présent dans l'arbre de gène T', et que le vecteur de bipartitions associé à l'arête (x,x_1) dans l'arbre d'espèces transformé T_1 (figure 2.8) est présent dans la table de bipartitions de T', alors le THG de (x,y) à (z,w), transformant T en T_1, fait partie d'***un scénario de coût minimal** *qui transforme T en T' et qui satisfait la contrainte de sous-arbres.*

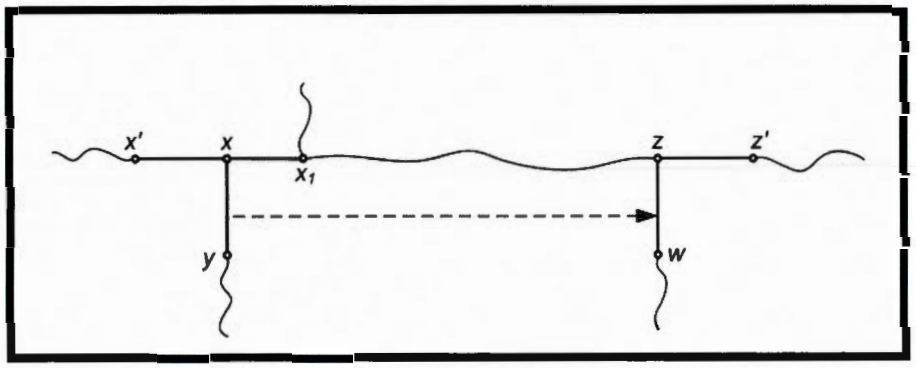


Figure 2.8 - Le transfert entre les arêtes (x,y) et (z,y) fait partie du scénario de coût minimal. Un scénario de coût minimal transforme l'arbre d'espèces T en l'arbre de gène T' si la bipartition correspondante à la branche (x,x_1) dans l'arbre transformé T_1 est présente dans la table de bipartitions T' et le sous-arbre Sub_{yw} est présent dans T'; (Théorème 2) tout scénario de coût minimum transforme l'arbre d'espèces T en l'arbre de gène T' si toutes les bipartitions correspondantes aux branches du chemin (x',z') dans l'arbre d'espèces T_1 sont présentes dans la table de bipartitions de T' et le sous-arbre Sub_{yw} est présent dans l'arbre T' (figure tirée de Boc *et al.*, 2010).

Théorème 2. *Si le sous-arbre Sub_{yw} nouvellement formé et résultant du THG (i.e. le sous-arbre enraciné par l'arête (x,a) sur la figure 2.5) est présent dans l'arbre de gène T', et que tous les vecteurs de bipartitions associés aux arêtes du chemin (x',z') dans l'arbre d'espèces transformé T_1 (figure 2.8) sont présents dans la table de bipartitions de T', et que le chemin (x',z') dans T_1 comprend au moins 3 arêtes, alors le THG de (x,y) à (z,w), transformant T en T_1, fait partie de* ***n'importe quel scénario de THGs de coût minimal*** *transformant T en T' et satisfaisant la contrainte de sous-arbres.*

Pour les démonstrations de ces théorèmes, le lecteur est référé à Boc *et al.* (2010). Les principales étapes de l'algorithme (voir la Figure 2.9), permettant de fournir une série de transformations ou d'opérations SPR de coût minimal d'un arbre d'espèces donné en un arbre de gène donné et basé sur ces deux théorèmes, sont les suivantes :

Étape préliminaire. Inférer les arbres phylogénétiques d'espèces et de gène, notés respectivement T et T'. Les feuilles de T et T' sont étiquetées par le même ensemble de n éléments (*i.e.*, d'espèces). Les deux arbres phylogénétiques doivent être enracinés en fonction des évidences biologiques existantes. Si aucune connaissance permettant d'enraciner les arbres n'est disponible, des stratégies d'enracinement par *outgroup* (en utilisant un groupe des espèces extérieur au groupe étudié) ou par *midpoint* (en utilisant l'arête médiane de l'arbre) peuvent être considérées. L'enracinement correct des arbres est primordial car un mauvais positionnement de la racine dans l'arbre d'espèces ou de gène peut fortement mener à l'inférence de transferts étant faux positifs ou faux négatifs. S'il existe dans T et T' des sous-arbres identiques ayant au moins 2 feuilles, réduire la taille du problème en remplaçant dans T et T' les sous-arbres identiques par les mêmes éléments auxiliaires.

Étape 1..k. Considérer l'ensemble des transferts possibles entre les paires d'arêtes de l'arbre d'espèces T_{k-1} ($T_0 = T$ à l'étape 1), sauf les transferts entre les paires d'arêtes adjacentes et ceux qui violent la contrainte de sous-arbres :

- Entre tous les THGs éligibles, rechercher les transferts qui satisfont les conditions du Théorème 2 d'abord et du Théorème 1 par la suite.
- Effectuer les opérations SPR correspondant aux transferts qui transforment l'arbre T_{k-1} en l'arbre T_k. Si de tels THGs n'existent pas, effectuer toutes les transformations SPR correspondant aux transferts satisfaisant la contrainte de sous-arbres.
- À chaque étape, de multiples mouvements SPR peuvent être effectués. La direction de chaque THG peut être déterminée en utilisant le critère d'optimisation sélectionné : les moindres carrés (MC), la distance de Robinson et Foulds (RF), la distance de quartets (QD) ou la dissimilarité de bipartitions (BD). Entre deux THGs opposés, l'algorithme choisit le transfert qui minimise la valeur du critère d'optimisation sélectionné, calculé pour l'arbre d'espèces transformé T_k et l'arbre de gène T'.

- Réduire ensuite la taille du problème en remplaçant des sous-arbres identiques ayant au moins 2 feuilles dans l'arbre d'espèces transformé T_k et l'arbre de gène T'.

La procédure s'arrête quand le coefficient MC, RF, QD ou BD est égal à 0. Grace à la procédure de réduction progressive de la taille des arbres d'espèces et de gène et de la possibilité de détecter plusieurs transferts à chaque étape, la complexité temporelle de l'algorithme proposé est $O(kn^4)$ pour inférer k transferts servant à réconcilier une paire de phylogénies d'espèces et de gène avec n feuilles. Cependant, dans la plupart des cas pratiques, cette complexité est $O(kn^3)$; pour plus de détails voir Boc *et al.* (2010).

Une fois l'arbre d'espèces et l'arbre de gène sont réconciliés, une procédure d'élimination des *transferts inutiles* est appliquée.

L'algorithme de détection de THGs complets selon Boc *et al.* (2010), qui est la version améliorée de l'algorithme que nous avons décrit dans la section 2.2, se présente comme suit (Figure 2.9). Dans la section 2.6, nous présentons le profilage du programme *HGT-Detection* implémentant cet algorithme amélioré afin de déterminer les parties de l'algorithme qui sont les plus intéressantes du point de vue de l'optimisation.

DÉBUT

Inférer les arbres d'espèces T et de gène T' sur le même ensemble d'espèces (i.e., feuilles);

Enraciner les arbres T et T' selon des évidences biologiques existantes ou en utilisant un *outgroup* ou un *point médian*;

Si (il existe des sous-arbres identiques, ayant au moins deux feuilles, entre T et T') alors

Diminuer la taille du problème en contractant les sous-arbres identiques dans T et T';

Sélectionner le critère d'optimisation *CO* entre : MC (moindres carrés), RF (distance de Robinson et Foulds), **DQ (distance de quartets)** ou DB (dissimilarité de bipartitions);

Calculer la valeur initiale du critère d'optimisation *CO* entre T et T';

$T_0 = T$;

$k = 0$; //k est l'indice de l'étape

Tant que ($CO \neq 0$) {

Trouver **l'ensemble des *THGs* candidats** (*i.e.*, obtenus par des opérations SPR) dans l'arbre T_k et représenté par E_THG_k;

L'ensemble E_THG_k contient seulement les transferts satisfaisant la contrainte de sous-arbres;

Tant que (il existe des *THGs* satisfaisant les conditions des Théorèmes 1 et 2) {

Si (il existe *THGs* $\in$ E_THG_k et satisfaisant les conditions du Théorème 2) alors

Effectuer les mouvements SPR correspondant à ces *THGs*;

Si (il existe *THGs* $\in$ E_THG_k et satisfaisant les conditions du Théorème 1) alors

Effectuer les mouvements SPR correspondant à ces *THGs*;

}

Effectuer tous les mouvements SPR restants correspondant aux *THGs* satisfaisant la contrainte de sous-arbres;

Recalculer la valeur du critère d'optimisation *CO* après l'ajout de chaque *THG*;

$k = k + 1$;

Diminuer la taille du problème en contractant les sous-arbres identiques dans T_k et T';

}

Éliminer les transferts inutiles obtenus;

FIN

Figure 2.9 - Algorithme de détection de transferts horizontaux de gènes complets selon Boc *et al.* (2010). Les améliorations par rapport à notre algorithme, présenté dans la section 2.2, sont indiquées en gras.

2.4 Version Interactive de l'algorithme de détection de transferts horizontaux de gènes complets

Étant donné que la détection de transferts horizontaux de gènes se fait automatiquement, sans connaissances *a priori* sur les données biologiques traitées, nous avons mis en place un processus de validation des transferts détectés afin d'augmenter la plausibilité des résultats de l'algorithme. Ainsi, la *version interactive* de l'algorithme que nous proposons ici, permet de valider, de modifier ou d'annuler les transferts détectés à chaque tour de boucle de l'algorithme *HGT-Detection* (Boc *et al.*, 2010). L'option de validation permet de confirmer le transfert détecté. L'option de modification permet de changer la direction du transfert détecté. Notons que la direction d'un transfert peut influencer grandement la valeur du critère d'optimisation choisi. Finalement, l'option d'annulation de transfert permet d'éliminer un transfert détecté.

La version interactive de l'algorithme de détection de THGs complets se présente comme suit (voir la Figure 2.10) :

DÉBUT

Inférer les arbres d'espèces T et de gène T' sur le même ensemble d'espèces (i.e., feuilles);

Enraciner les arbres T et T' selon des évidences biologiques existantes ou en utilisant un *outgroup* ou un *point médian*;

Si (il existe des sous-arbres identiques, ayant au moins deux feuilles, entre T et T') alors

Diminuer la taille du problème en contractant les sous-arbres identiques dans T et T';

Sélectionner le critère d'optimisation *CO* entre : MC (moindres carrés), RF (distance de Robinson et Foulds), DQ (distance de quartets) ou DB (dissimilarité de bipartitions);

Insérer les *transferts initiaux* suggérés par l'utilisateur (voir la Figure 2.11);

Calculer la valeur initiale du critère d'optimisation *CO* entre T et T';

$T_0 = T$;

$k = 0$; //k est l'indice de l'étape

Tant que ($CO \neq 0$ ou l'ajout de THG est impossible à cause des contraintes biologiques) {

Trouver l'ensemble des *THGs* candidats (*i.e.*, obtenus par des opérations SPR) dans l'arbre T_k et représenté par E_THG_k;

L'ensemble E_THG_k contient seulement les transferts satisfaisant la contrainte de sous-arbres;

Tant que (il existe des *THGs* satisfaisant les conditions des Théorèmes 1 et 2) {

Si (il existe $THGs \in E\_THG_k$ et satisfaisant les conditions du Théorème 2) alors

Effectuer les mouvements SPR correspondant à ces *THGs*;

Valider et/ou modifier les transferts détectés à cette étape (voir la Figure 2.12);

Si (il existe $THGs \in E\_THG_k$ et satisfaisant les conditions du Théorème 1) alors

Effectuer les mouvements SPR correspondant à ces *THGs*;

Valider et/ou modifier les transferts détectés à cette étape (voir la Figure 2.12);

}

Effectuer tous les mouvements SPR restants correspondant aux *THGs* satisfaisant la contrainte de sous-arbres;

Valider et/ou modifier les transferts détectés restants (voir la Figure 2.12);

Recalculer la valeur du critère d'optimisation *CO* après l'ajout de chaque *THG*;

$k = k + 1$;

Diminuer la taille du problème en contractant les sous-arbres identiques dans T_k et T';

}

Éliminer les transferts inutiles obtenus;

FIN

Figure 2.10 - Version interactive de l'algorithme de détection de transferts horizontaux de gènes. Les modifications par rapport à l'algorithme *HGT-Detection*, présenté sur la Figure 2.9, sont indiquées en gras.

L'insertion des transferts initiaux (voir la Figure 2.11) permet d'incorporer à l'arbre d'espèces des transferts liés à des informations biologiques pertinentes sur les espèces traitées.

DÉBUT

Tant que (les *THGs* initiaux suggérés satisfont les contraintes d'évolution voir figure 2.4)

{

Effectuer les mouvements SPR correspondant à ces *THGs*;

}

Recalculer la valeur du critère d'optimisation *CO* après l'ajout des transferts initiaux;

FIN

Figure 2.11 - Procédure d'introduction des transferts horizontaux de gènes initiaux.

Un tel dispositif de validation (voir la Figure 2.12) permet d'avoir un plus grand contrôle au niveau des transferts détectés car des transferts considérés comme valides, selon le modèle mathématique mis en place, peuvent être considérés comme erronés selon les connaissances biologiques disponibles. Il est important de souligner que les transferts détectés à chaque tour de la boucle principale de l'algorithme ont une grande influence sur les transferts subséquents.

DÉBUT

Sélectionner le *THG* ∈ *E_ THG* proposé par l'algorithme;
Choisir l'option : *Confirmer*, *Annuler* ou *Inverser* le transfert;
Si l'option choisie est (*Confirmer*) alors
    Le *THG* proposé par l'algorithme est validé;
Sinon si l'option choisie est (*Annuler*) alors
    Effectuer une opération correspondant au retrait du *THG* proposé par l'algorithme;
Sinon si l'option choisie est (*Inverser*) alors
    Effectuer une opération correspondant au retrait du *THG* proposé par l'algorithme;
    Effectuer une opération correspondant à l'ajout du *THG* inverse;
Mettre à jour la valeur du critère d'optimisation *CO*;

FIN

Figure 2.12 - Procédure de validation des transferts horizontaux de gènes détectés dans la version interactive de l'algorithme.

Si aucune modification n'est effectuée après tous les tours de la boucle de recherche des transferts, les résultats fournis par la version interactive sont identiques à ceux de la version séquentielle. La procédure est exécutée jusqu'à ce que le coefficient RF, MC, QD ou BD soit égal à 0 ou que l'ajout de THG est impossible à cause des contraintes biologiques (voir la Figure 2.4). La complexité temporelle de la version interactive de l'algorithme est $O(kn^4)$ pour inférer k transferts servant à réconcilier une paire de phylogénies d'espèces et de gène avec n feuilles. Toutefois, comme dans le cas de l'algorithme *HGT-Detection*, cette complexité est $O(kn^3)$ dans la plupart des cas pratiques.

2.5 Version Consensus de l'algorithme de détection de transferts horizontaux de gènes complets

Les arbres de consensus ont été bien étudiés et très utilisés en analyse phylogénétique (Nelson, 1979; Margush et McMorris, 1981). Cependant, aucune méthode de consensus n'a été proposée pour réconcilier différents scénarios des transferts horizontaux de gènes. Nous proposons ici une telle méthode. Dans le cas de la *version consensus* de l'algorithme de

détection de THGs, nous avons besoin d'un arbre d'espèces unique et d'un ensemble d'arbres de gène qui sont inférés à partir des séquences répliquées, par exemple. Tous les transferts apparaissant dans tous les scénarios obtenus pour cet ensemble d'arbres de gène, sont identifiés et comptés. Cette identification est effectuée en comparant les opérations SPR correspondantes.

Dans cette étude, deux THGs (ou deux opérations SPR) sont considérés comme identiques si et seulement si *les bipartitions des deux arêtes donneuse et receveuse sont équivalentes pour les deux transferts*. Une autre solution, plus stricte, serait de considérer que ces deux transferts sont identiques si et seulement si *les sous-arbres donneur et receveur sont identiques pour les deux transferts*. Comme les données considérées sont souvent des réplicats des mêmes séquences, cette deuxième stratégie produirait dans la plupart des cas de plus faibles scores de consensus de THG, particulièrement pour des phylogénies mal résolues. Dans les arbres de gène, la racine peut être placée de la façon suivante : si une arête induisant une bipartition identique à l'arête de la racine dans l'arbre d'espèces n'existe pas dans l'arbre de gène répliqué, alors la racine peut être placée sur une arête produisant la bipartition la plus proche de celle de l'arbre d'espèces. On affiche à la fin les transferts dont le score de consensus est supérieur ou égal au seuil défini par l'utilisateur.

La *version consensus* de l'algorithme de détection de THGs complets se présente comme suit (voir la Figure 2.13) :

DÉBUT

Enraciner l'arbre d'espèces T selon des évidences biologiques existantes ou en utilisant un *outgroup* ou un *point médian*;

Pour chaque arbre de gène $T'_i \in$ l'ensemble des arbres de gènes (indice i allant de 0...n) {

Enraciner l'arbre de gène T'_i selon des évidences biologiques existantes ou en plaçant la racine sur une arête produisant la bipartition la plus proche de l'arête de la racine de l'arbre d'espèces T;

Si (il existe des sous-arbres identiques, ayant au moins deux feuilles, entre T et T'_i) alors

Diminuer la taille du problème en contractant les sous-arbres identiques dans T et T'_i;

Sélectionner le critère d'optimisation *CO* entre : MC (moindres carrés), RF (distance de Robinson et Foulds), DQ (distance de quartets) ou DB (dissimilarité de bipartitions);

Calculer la valeur initiale du critère d'optimisation *CO* entre T et T'_i; $T_0 = T$; $k = 0$;

Tant que ($CO \neq 0$) {

Trouver l'ensemble des *THGs* candidats (*i.e.*, obtenus par des opérations SPR) dans l'arbre T_k et représenté par E_THG_k;

L'ensemble E_THG_k contient seulement les transferts satisfaisant la contrainte de sous-arbres;

Tant que (il existe des *THGs* satisfaisant les conditions des Théorèmes 1 et 2) {

Si (il existe $THGs \in E\_THG_k$ et satisfaisant les conditions du Théorème 2) alors

Effectuer les mouvements SPR correspondant à ces *THGs*;

Si (il existe $THGs \in E\_THG_k$ et satisfaisant les conditions du Théorème 1) alors

Effectuer les mouvements SPR correspondant à ces *THGs*;

}

Effectuer tous les mouvements SPR restants correspondant aux *THGs* satisfaisant la contrainte de sous-arbres;

Recalculer la valeur du critère d'optimisation *CO* après l'ajout de chaque *THG*;

$k = k + 1$;

Diminuer la taille du problème en contractant les sous-arbres identiques dans T_k et T'_i;

}

Éliminer les transferts inutiles obtenus;

Mettre à jour la liste des transferts détectés en y ajoutant les nouveaux transferts;

Recalculer les scores de consensus pour chaque transfert de la liste;

Éliminer les transferts dont le score de consensus n'atteindra jamais le seuil défini;

$i = i + 1$;

}

Afficher les transferts de consensus en fonction du seuil défini;

FIN

Figure 2.13 - Version consensus de l'algorithme de détection de transferts horizontaux de gènes. Les modifications par rapport à l'algorithme *HGT-Detection*, présenté sur la Figure 2.9, sont indiquées en gras.

La mise à jour de la liste des transferts ainsi que son décompte n'est pas un processus trivial. Ce processus implique la recherche des associations d'équivalences entre les sous-arbres des différents arbres de gène pour détecter les transferts équivalents impliquant les sous-arbres donneur et receveur identiques. À chaque étape, nous recalculons tous les scores de consensus pour chaque transfert de la liste et puis nous éliminons les transferts dont le score de consensus n'atteindra jamais le seuil de consensus défini.

La complexité temporelle de la version consensus de l'algorithme est $O(rkn^4)$ pour inférer k transferts horizontaux servant à réconcilier un arbre d'espèces avec un ensemble de r arbres de gène, ayant tous n feuilles.

2.6 Profilage du programme de détection de transferts horizontaux de gènes complets *HGT-Detection*

L'opération de profilage consiste à analyser un programme afin d'en connaitre certaines caractéristiques, comme par exemple la liste des fonctions appelées et le temps passé dans chacune de ces fonctions. Le profilage permet d'identifier les parties du code d'un programme que nous pouvons optimiser, à travers la parallélisation entre autres. Étant donné que le programme de détection de THGs complets a été développé dans le langage *C*, nous avons effectué l'opération de profilage à travers la commande *gprof* de *Linux* (sur le cluster *T-Rex* installé au laboratoire *LAMISS* de l'UQAM; le système d'exploitation utilisé était *CentOS release 5.7 (Final)*). Cette commande produit des rapports de performances de base du programme. Lors de l'opération de profilage, nous avons utilisé comme paramètres d'entrée un arbre d'espèces et un arbre de gène aléatoires contenant 320 feuilles chacun.

Les résultats détaillés du profilage du programme de détection de THGs complets sont présentés ci-dessous, dans le Tableau 2.1.

% de Temps	Temps cumulatif / (en sec)	Temps individuel (en sec)	Nombre d'appels de la fonction	Temps moyen par appel (en ms)	Temps total moyen par appel (en ms)	Noms de la fonction
53.92	680.76	680.76	185503	0.00	0.00	**copyInputTree**(InputTree*, InputTree, int, int)
34.62	1117.85	437.08	67	6.52	6.52	**Floyd**(double**, double**, int, int)
4.88	1179.43	61.58	53	1.16	1.16	**approx_arb**(double**,double**,double**, double**,int*,long*,double*,int,int*,int,int)
2.48	1210.78	31.35	4	7.84	7.84	**Floyd**(double**, double**, double**, int, int)
2.45	1241.74	30.96	1527	0.02	0.02	**BipartitionDistance**(int**, int**, int)
0.59	1249.20	7.46	746	0.01	0.02	**applyHGT**(double**, InputTree*, int, int)
0.55	1256.10	6.90	817	0.01	0.01	**loadAdjacenceMatrix**(double**, long*, double*, int, int)
0.20	1258.64	2.54	183823	0.00	0.00	**TestSubTreeConstraint**(InputTree, int, int, DescTree*, DescTree*)
0.06	1259.41	0.77	3362	0.00	0.00	**Bipartition_Table**(double**, int**, int*, int)
0.05	1260.03	0.62	3376	0.00	0.00	**odp1**(double**, int*, int*, int*, int)
0.03	1260.41	0.38	25131960	0.00	0.00	**vecteursEgaux**(DescTree, DescTree)
0.03	1260.73	0.32	262177	0.00	0.00	**isAValidHGT**(InputTree, int, int)
0.02	1261.02	0.29	25	0.01	7.71	**applyHGT2**(double**,InputTree*, int,int)
0.02	1261.25	0.23	1681	0.00	0.00	**Table_Comparaison**(int**, int**,int*,int*,int,int,int)
0.02	1261.46	0.21	2	0.11	0.11	**NJ**(double**, double**, int)
0.02	1261.65	0.19	19	0.01	0.02	**RechercherBipartition**(long*, double**,int,double**,DescTree*,int, int)
0.01	1261.81	0.16	1527	0.00	0.02	**computeCriteria**(double**, double**, int, CRITERIA*, double*, long*, double*, long*)
0.01	1261.96	0.15	6	0.03	115.43	**findBestHGTtab**(InputTree, InputTree, Parameters, HGT*, int*, int*, int)
0.01	1262.09	0.13	N/D	N/D	N/D	**main**
0.01	1262.18	0.09	14341	0.00	0.00	**findFils**(double**, int, int)

0.00	1262.24	0.06	6	0.01	17.51	**ReduceTree**(InputTree, InputTree, InputTree*, InputTree*, ReduceTrace*, DescTree*, DescTree*, int,int)
0.00	1262.27	0.03	56328	0.00	0.00	**TrierTableau**(int*, int)
0.00	1262.30	0.03	14	0.00	6.54	**CreateSubStructures**(InputTree*, int, int)
0.00	1262.32	0.02	1510	0.00	0.00	**findBranch**(InputTree, int*, int*)
0.00	1262.34	0.02	14	0.00	0.00	**Tree_edges**(double**, long*, double*, int, int)
0.00	1262.36	0.02	2	0.01	0.01	**lectureNewick**(char const*, long*, double*, char**, int*)
0.00	1262.38	0.02	2	0.01	7.86	**addRoot**(InputTree*, InputTree*, char const*, char const*, char*, char*, int*, char const*)
0.00	1262.40	0.02	N/D	N/D	N/D	**UpdateCriterion**(int*,char const*,CRITERIA, HGT*, int, int, int)
0.00	1262.41	0.01	3590	0.00	0.00	**sortIntTab**(int*, int, int)
0.00	1262.42	0.01	320	0.00	0.00	**ListeSommets_taille_0**(double**, int*, int)
0.00	1262.43	0.01	19	0.00	0.00	**viderArbre**(TNoeud*)
0.00	1262.44	0.01	2	0.01	0.01	**readInput**(int, char const*, InputTree*)
0.00	1262.45	0.01	1	0.01	0.01	**TrierMatrices**(double**, char**, char**, int)
0.00	1262.46	0.01	1	0.01	0.01	**ajouterMatriceGene**(double**, char const*, int, char**)
0.00	1262.47	0.01	N/D	N/D	N/D	**PrintHeader**(_IO_FILE*, Parameters)
0.00	1262.48	0.01	N/D	N/D	N/D	**frame_dummy**
0.00	1262.48	0.00	143178	0.00	0.00	**isInside**(DescTree, DescTree)
0.00	1262.48	0.00	638	0.00	0.00	**xtoa**(unsigned long, char*, unsigned int, int)
0.00	1262.48	0.00	638	0.00	0.00	**itoa_**(int, char*, int)
0.00	1262.48	0.00	380	0.00	0.00	**deleteSommet**(DescTree*, int, int)
0.00	1262.48	0.00	154	0.00	0.00	**TestSubTreeLeafs**(InputTree, int, int, DescTree*, DescTree*)
0.00	1262.48	0.00	130	0.00	0.00	**loadCriteria**(CRITERIA, HGT*)
0.00	1262.48	0.00	40	0.00	0.00	**TestCriterionAndUpdate**(int*, char const*, CRITERIA, HGT*, int, int, int, int)
0.00	1262.48	0.00	33	0.00	0.00	**InitCriteria**(CRITERIA*, int)
0.00	1262.48	0.00	32	0.00	0.00	**initInputTree**(InputTree*)

0.00	1262.48	0.00	30	0.00	0.00	**expandBestHGT**(HGT, HGT*, ReduceTrace, DescTree*, InputTree)
0.00	1262.48	0.00	30	0.00	0.00	**findListSpecies**(HGT*, DescTree*, InputTree)
0.00	1262.48	0.00	28	0.00	7.69	**AdjustBranchLength**(InputTree*, InputTree, int, int)
0.00	1262.48	0.00	20	0.00	0.00	**FreeCriteria**(CRITERIA*, int)
0.00	1262.48	0.00	20	0.00	0.00	**printTransfer**(_IO_FILE*, int, char**, int, int, int, int, int)
0.00	1262.48	0.00	20	0.00	0.00	**copyHGT**(HGT, HGT*)
0.00	1262.48	0.00	19	0.00	0.00	**ParcoursArbre**(TNoeud*, DescTree*)
0.00	1262.48	0.00	19	0.00	0.00	**CreerSousArbre**(long*, int*, double**, int, int)
0.00	1262.48	0.00	19	0.00	0.00	**deleteBipartition**(DescTree*, InputTree)
0.00	1262.48	0.00	17	0.00	0.00	**allocMemmory**(InputTree*, int)
0.00	1262.48	0.00	12	0.00	0.00	**FreeMemory_InputTreeReduced**(InputTree*, int)
0.00	1262.48	0.00	11	0.00	0.00	**FreeMemory_InputTree**(InputTree*, int)
0.00	1262.48	0.00	2	0.00	0.00	**readNewick**(_IO_FILE*)
0.00	1262.48	0.00	2	0.00	7.86	**newickToMatrix**(char const*, InputTree*)
0.00	1262.48	0.00	2	0.00	0.00	**nbSpeciesNewick**(char const*)
0.00	1262.48	0.00	2	0.00	7.85	**addLeafAndUpdate**(InputTree*, int)
0.00	1262.48	0.00	2	0.00	0.00	**nextTreeIsNewick**(_IO_FILE*)
0.00	1262.48	0.00	2	0.00	0.00	**printRootByLeaves**(char*, int, InputTree*)
0.00	1262.48	0.00	2	0.00	0.00	**midPoint**(long*, double**, int, int)
0.00	1262.48	0.00	2	0.00	0.00	**printRoot**(char*, int, int)
0.00	1262.48	0.00	1	0.00	0.00	**formatResult**(HGT*, int, HGT*, InputTree)
0.00	1262.48	0.00	1	0.00	0.00	**ecrireMatrice**(double**, char const*, int, char**)
0.00	1262.48	0.00	1	0.00	15.72	**readInputFile**(_IO_FILE*, char const*, char*)
0.00	1262.48	0.00	1	0.00	0.00	**filtrerMatrice**(double**, double**, char**, char**, int, int, char*)
0.00	1262.48	0.00	1	0.00	0.00	**readParameters**(Parameters*, char**, int)

0.00	1262.48	0.00	1	0.00	0.00	**ExtraireDonnees**(char const*, char*, char*)
0.00	1262.48	0.00	1	0.00	203.92	**DeleteUseLessHGT**(int, HGT*, InputTree, InputTree)
0.00	1262.48	0.00	1	0.00	0.00	**supprimerSousEnsemble**(DescTree*, int)
0.00	1262.48	0.00	1	0.00	0.00	**printHGT**(_IO_FILE*, CRITERIA*, char*, int, InputTree, HGT*, int, double*, char*, int)

*sec : seconde

*ms : milliseconde

Tableau 2.1 - Tableau de toutes les fonctions du programme de détection de transferts horizontaux complets de gènes, *HGT-Detection*.

Le Tableau 2.1 fournit des informations sur les fonctions du programme *HGT-Detection*. Plus précisément, les statistiques suivantes sont présentées :

- Le pourcentage de temps (% de temps) : représente le pourcentage de la durée totale du programme utilisé par la fonction.
- Le temps cumulatif (en secondes) : représente le total du temps passé dans cette fonction et du temps passé dans les fonctions qui sont énumérées avant elle.
- Le temps individuel (en secondes) : représente le temps total passé dans la fonction.
- Le nombre d'appels : représente le nombre de fois que la fonction a été invoquée.
- Le temps moyen par appel (en millisecondes) : représente le temps moyen passé dans la fonction.
- Le temps total moyen par appel (en millisecondes): représente le temps moyen passé dans la fonction et dans ses fonctions descendantes.
- Le nom : représente le nom de la fonction.

En analysant le Tableau 2.1, nous remarquons que les cinq premières fonctions : *copyInputTree*(*InputTree**, *InputTree*, *int*, *int*), *Floyd*(*double***, *double***, *int*, *int*), *approx_arb*(*double***, *double***, *double***, *double***, *int**, *long**, *double**, *int*, *int**, *int*, *int*), *Floyd*(*double***, *double***, *double***, *int*, *int*) et *BipartitionDistance*(*int***, *int***, *int*) totalisent 98% du temps total du programme. Notons que les deux premières fonctions

mentionnées, qui totalisent 88.5% du temps total, sont des fonctions très courtes dont la parallélisation ne s'avérera pas très efficace. Cependant, dans la plupart des cas, ces cinq fonctions sont appelées à partir de la fonction *findBestHGTtab(InputTree, InputTree, Parameters, HGT*, int*, int*, int)* qui contient la boucle principale de la recherche des transferts du programme.

Notons aussi que la fonction *findBestHGTtab(InputTree, InputTree, Parameters, HGT*, int*, int*, int)* est le deuxième en fonction du temps total moyen par appel (avec 115.45 millisecondes). La fonction qui consomme le plus de temps total moyen par appel est *DeleteUseLessHGT(int, HGT*, InputTree, InputTree)* qui totalise 203.92 millisecondes. Les sept fonctions mentionnées sont les plus intéressantes du point de vue d'une optimisation.

Le Tableau 2.2 décrit l'arbre d'appel du programme *HGT-Detection*, fourni également à travers la commande *gprof*. Ce tableau a été trié en fonction du temps total passé dans chaque fonction et dans ses fonctions descendantes. Le Tableau 2.3, quant à lui, nous donne plus de détails sur l'exécution de la fonction *findBestHGTtab(InputTree, InputTree, Parameters, HGT*, int*, int*, int)*. En analysant ces deux tableaux, nous voyons clairement que la fonction *findBestHGTtab(InputTree, InputTree, Parameters, HGT*, int*, int*, int)* totalise presque 55% du temps total du programme.

Indice	Pourcentage de temps	Temps individuel (en sec)	Temps des descendants (en ms)	Nombre d'appels	Nom de la fonction
[1]	100.0	0.13	1262.31		main [1]
		0.15	692.41	6/6	**findBestHGTtab**(InputTree, InputTree, Parameters, HGT*, int*, int*, int) [2]
		0.00	203.92	1/1	**DeleteUseLessHGT**(int, HGT*, InputTree, InputTree) [6]
		0.29	192.35	25/25	**applyHGT2**(double**, InputTree*, int, int) [7]
		0.06	105.00	6/6	**ReduceTree**(InputTree, InputTree, InputTree*, InputTree*, ReduceTrace*, DescTree*,DescTree*, int, int) [8]
		0.00	23.08	3/28	**AdjustBranchLength**(InputTree*, InputTree, int, int) [5]
		0.00	15.72	1/1	**readInputFile**(_IO_FILE*, char const*, char*) [14]

		0.02	15.69	2/2	**addRoot**(InputTree*, InputTree*, char const*, char const*, char*, char*, int*, char const*) [16]
		0.00	13.07	2/14	**CreateSubStructures**(InputTree*, int, int) [9]
		0.21	0.00	2/2	**NJ**(double**, double**, int) [27]
		0.00	0.17	8/1527	**computeCriteria**(double**, double**, int, CRITERIA*, double*, long*, double*, long*) [11]
		0.07	0.04	7/19	**RechercherBipartition**(long*, double**, int, double**, DescTree*, int, int) [25]
		0.03	0.00	7/185503	**copyInputTree**(InputTree*, InputTree, int, int) [3]
		0.01	0.00	320/320	**ListeSommets_taille_0**(double**, int*, int) [37]
		0.01	0.00	2/2	**readInput**(int, char const*, InputTree*) [40]
		0.01	0.00	1/1	**TrierMatrices**(double**, char**, char**, int) [41]
		0.00	0.00	30/30	**expandBestHGT**(HGT, HGT*, ReduceTrace, DescTree*, InputTree) [44]
		0.00	0.00	17/32	**initInputTree**(InputTree*) [58]
		0.00	0.00	12/12	**FreeMemory_InputTreeReduced**(InputTree*, int) [65]
		0.00	0.00	7/19	**deleteBipartition**(DescTree*, InputTree) [63]
		0.00	0.00	4/11	**FreeMemory_InputTree**(InputTree*, int) [66]
		0.00	0.00	2/33	**InitCriteria**(CRITERIA*, int) [57]
		0.00	0.00	1/1	**readParameters**(Parameters*, char**, int) [76]
		0.00	0.00	1/1	**formatResult**(HGT*, int, HGT*, InputTree) [73]
		0.00	0.00	1/1	**printHGT**(_IO_FILE*, CRITERIA*, char*, int, InputTree, HGT*, int, double*, char*, int) [79]
		0.00	0.00	1/20	**FreeCriteria**(CRITERIA*, int) [60]

Tableau 2.2 - Tableau représentant l'arbre d'appel du programme *HGT-Detection*.

Indice	Pourcentage de temps	Temps individuel (en sec)	Temps des descendants (en ms)	Nombre d'appels	Nom de la fonction
		0.15	**692.41**	**6/6**	**main [1]**
[2]	**54.9**	**0.15**	**692.41**	**6**	**findBestHGTtab(InputTree, InputTree, Parameters, HGT*, int*, int*, int) [2]**
		680.64	0.00	185471/185503	**copyInputTree**(InputTree*, InputTree, int, int) [3]
		0.03	5.29	249/1527	**computeCriteria**(double**, double**, int, CRITERIA*, double*, long*, double*, long*) [11]
		2.54	0.56	183823/183823	**TestSubTreeConstraint**(InputTree, int, int, DescTree*, DescTree*) [20]
		1.54	1.30	154/746	**applyHGT**(double**, InputTree*, int, int) [18]
		0.32	0.00	261452/262177	isAValidHGT(InputTree, int, int) [24]
		0.12	0.07	12/19	**RechercherBipartition**(long*, double**, int, double**, DescTree*, int, int) [25]
		0.00	0.00	154/154	**TestSubTreeLeafs**(InputTree, int, int, DescTree*, DescTree*) [45]
		0.00	0.00	130/130	**loadCriteria**(CRITERIA, HGT*) [55]
		0.00	0.00	40/40	**TestCriterionAndUpdate**(int*, char const*, CRITERIA, HGT*, int, int, int, int) [56]
		0.00	0.00	30/30	**findListSpecies**(HGT*, DescTree*, InputTree) [59]
		0.00	0.00	24/33	**InitCriteria**(CRITERIA*, int) [57]
		0.00	0.00	12/32	**initInputTree**(InputTree*) [58]
		0.00	0.00	12/19	**deleteBipartition**(DescTree*, InputTree) [63]
		0.00	0.00	12/20	**FreeCriteria**(CRITERIA*, int) [60]
		0.00	0.00	6/11	**FreeMemory_InputTree**(InputTree*, int) [66]

*sec : seconde

*ms : milliseconde

Tableau 2.3 - Tableau représentant l'arbre d'appel de la fonction *findBestHGTtab*.

Pour les fonctions en cours (voir la fonction *main* dans le Tableau 2.2 et la fonction *findBestHGTtab* dans le Tableau 2.3), les champs ont les significations suivantes :

- L'indice : représente un numéro unique attribué à chaque fonction du programme.

- Le pourcentage du temps : représente le pourcentage de la durée totale du programme utilisé par la fonction et ses descendants.
- Le temps individuel (en secondes) : représente le temps total passé dans la fonction.
- Le temps des descendants (en millisecondes) : représente le temps total passé dans les descendants de la fonction.
- Le nombre d'appels : représente le nombre de fois que la fonction a été invoquée.
- Le nom : représente le nom de la fonction en cours.

Concernant les parents de la fonction (voir la fonction *main* dans le Tableau 2.3), les champs ont les significations suivantes :

- Le temps individuel : représente le temps passé dans la fonction après l'appel du parent.
- Le temps des descendants : représente le temps total passé dans les fonctions descendantes après l'appel du parent.
- Le nombre d'appels : représente le nombre de fois que le parent a appelé la fonction divisé par nombre total d'appels de la fonction.
- Le nom : représente le nom du parent.

Pour les descendants de la fonction (voir toutes les fonctions dans le Tableau 2.2, en excluant la fonction *main*, et toutes les fonctions du Tableau 2.3, en excluant les fonctions *main* et *findBestHGTtab*), les champs ont les significations suivantes :

- Le temps individuel : représente le temps passé par le descendant dans la fonction.
- Le temps des descendants: représente le temps total passé dans les descendants du descendant de la fonction.
- Le nombre d'appels : représente le nombre de fois que la fonction a appelé son descendant divisé par le nombre total de fois que le descendant a été appelé.
- Le nom représente le nom du descendant.

Le chapitre IV de la thèse traitera le sujet de la parallélisation du programme *HGT-Detection*, et plus précisément de la fonction *findBestHGTtab*, qui contient la boucle principale du programme de la recherche des THGs (*i.e.*, boucles *for* imbriquées; voir la ligne *Trouver l'ensemble des THGs candidats dans l'arbre T_k et représenté par E_THG_k;* sur la Figure 2.9).

2.7 Résumé

Dans ce chapitre, nous avons présenté trois nouveaux algorithmes de détection de transferts horizontaux de gènes complets : détection directe ou standard, détection interactive et détection consensus. Dans le chapitre suivant, nous présenterons et comparerons entre elles deux nouveaux algorithmes de détection de transferts horizontaux de gènes partiels.

Un article portant sur l'inférence des transferts horizontaux de gènes complets (Makarenkov *et al.*, 2007) a été rédigé et publié dans le cadre de ce projet doctoral. J'ai participé à la définition et à la description de la dissimilarité de bipartitions, ainsi qu'à la réalisation des tests de validation par simulations du critère de dissimilarité de bipartitions. J'ai proposés et implémenté l'algorithme de détection de THGs complets décrit dans la section 2.2 et les algorithmes interactif et consensus présentés dans les sections 2.4 et 2.5. J'ai aussi développé les interfaces Web (voir le chapitre V) pour ces trois algorithmes de détection de THGs complets présentés dans ce chapitre.

CHAPITRE III

ALGORITHMES POUR LA DÉTECTION DE TRANSFERTS HORIZONTAUX DE GÈNES PARTIELS

Dans ce chapitre, nous présenterons deux nouveaux algorithmes pour la détection de transferts horizontaux de gènes partiels menant à la formation des gènes mosaïques.

3.1 Introduction

Il est connu que les bactéries et les archées s'adaptent aux différentes conditions de leur environnement grâce à l'acquisition de gènes mosaïques (Gogarten *et al.*, 2002). Le terme « mosaïque » résulte de la configuration des blocs entrecoupés de séquences ayant des histoires d'évolution différentes. Cependant, ces blocs se retrouvent combinés dans l'allèle (*i.e.*, copie du gène) résultant suite à des évènements de la recombinaison (

Figure 3.1). Les segments recombinés peuvent être dérivés d'autres souches d'espèces (Hollingshead *et al.*, 2000, Gogarten *et al.*, 2002). Les gènes mosaïques correspondent au cas des transferts horizontaux partiels.

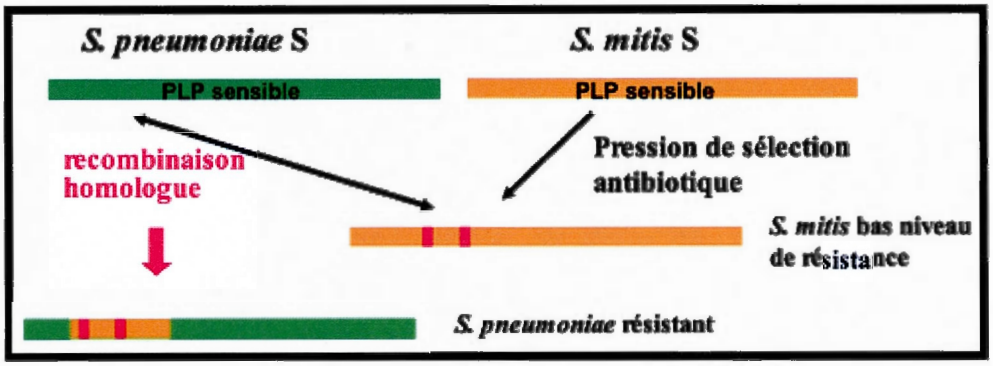


Figure 3.1 - Création d'un gène mosaïque par recombinaison (figure tirée du site Web de l'Université d'Angers).

Nous avons vu qu'il existe une diversité de méthodes proposées pour l'identification et la validation de THGs complets (Hein, 1990; von Haeseler et Churchill, 1993; Mirkin *et al.*, 1995, Maddison, 1997; Hallett et Lagergren, 2001; Boc et Makarenkov, 2003; MacLeod *et al.*, 2005; Nakhleh *et al.*, 2005; Beiko et Hamilton, 2006; Boc *et al.*, 2010). Toutefois, il existe seulement deux méthodes qui traitent du problème de l'inférence de THGs partiels et qui prédisent les origines des gènes mosaïques (Denamur *et al.*, 2000 et Makarenkov *et al.*, 2006). Malheureusement, ces deux travaux ne traitent pas du sujet de la validation des transferts partiels obtenus et n'incluent pas de simulations pour tester les performances des algorithmes dans différentes situations pratiques.

Le premier algorithme que nous proposons se base sur une optimisation par les moindres carrés pour évaluer les portions de séquences transférées. Un modèle d'optimisation et un algorithme seront alors présentés. Le deuxième algorithme est associé au principe de fenêtres coulissantes. Cette deuxième approche permet d'identifier des transferts partiels en se déplaçant le long d'un alignement de séquences multiples. Une procédure de bootstrap sera aussi utilisée pour évaluer le support de chaque transfert génétique prédit.

Considérons un exemple de transferts partiel présenté sur la Figure 1.8. L'arête en pointillé qui relie les arêtes (3,4) et (2,C) représente un transfert horizontal de gène inscrit dans l'arbre d'espèces (l'arbre du haut de la Figure 1.8). Dans un arbre phylogénétique, il existe toujours un chemin unique reliant toutes les paires de sommets. Lors d'un transfert

partiel de gènes, l'addition d'une arête qui représente le transfert horizontal, crée un autre chemin entre certains sommets, transformant ainsi l'arbre phylogénétique initial en réseau (Figure 1.8, l'arbre du bas à gauche). La distance évolutive entre l'espèce C et toute autre espèce dans ce réseau doit être calculée comme étant la somme des parties des longueurs des chemins reliant ces deux espèces. Ces chemins passent forcement soit par l'arête (6,2), soit par l'arête (6,7) (voir la Figure 1.8, l'arbre du bas à gauche). Plusieurs règles d'évolution doivent être incorporées dans ce modèle pour permettre une meilleure interprétation biologique. Parmi ces règles, nous avons : la définition du sens de l'évolution – de la racine vers les feuilles, l'interdiction de transferts entre les arêtes situées sur la même lignée, l'interdiction de plusieurs transferts croisés entre deux lignées données, etc. (voir la Figure 2.4).

Cependant, l'ajout des règles biologiques et la transformation de l'arbre phylogénétique en réseau font en sorte que le calcul des chemins dans le graphe orienté, obtenu après l'ajout des arêtes de transferts horizontaux, doit se faire par approximation. L'utilisation de formules d'approximation permet de calculer la longueur des chemins en temps polynomial. L'algorithme de détection de transferts partiels décrit dans Makarenkov *et al.* (2006) utilise l'optimisation par les moindres carrés pour déterminer les transferts de gènes les plus appropriés sous les contraintes biologiques définies.

Le modèle général du transfert partiel est illustré dans la Figure 3.2. Dans l'algorithme existant de transferts partiels de Makarenkov *et al.* (2006), le chemin de poids minimum entre les taxa (*i.e.*, espèces) i et j est modifié après l'ajout d'une nouvelle arête orientée (a,b), dirigée de b vers a, et représentant un THG partiel. D'un point de vue biologique, il est nécessaire d'envisager que le transfert horizontal de gène entre b et a affecte la distance évolutive entre le taxon i et le taxon j, dont la position dans l'arbre phylogénétique est fixe, si et seulement si i est situé au-dessous de l'arête de transfert. Par contre, la distance évolutive entre i_1 et j ne doit pas être affectée par ajout de l'arête (a,b).

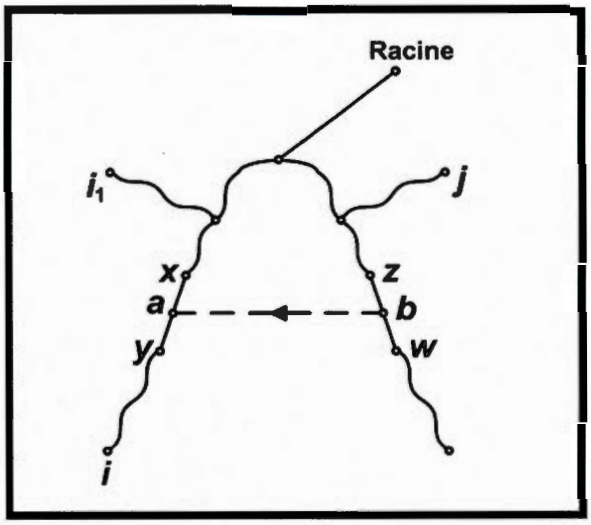


Figure 3.2 - Modèle d'évolution impliquant des transferts partiels (figure tirée de Makarenkov *et al.* 2008).

Makarenkov *et al.* (2006) ont défini l'ensemble $A(a,b)$ incluant toutes les paires de taxa telles que la taille minimum du chemin entre eux peut changer si un transfert (a,b) est ajouté à l'arbre d'espèces T. Le calcul de cet ensemble se fait selon la condition suivante :

$$Min\{d(i,a) + d(j,b);\ d(j,a) + d(i,b)\} < d(i,j),$$

où $d(i,j)$ est le chemin de taille minimum entre les taxa i et j dans T. Donc, $A(a,b)$ est l'ensemble de toutes les paires de feuilles ij telles que $dist(i,j) > 0$; $dist(i,j)$ indique la distance entre les nœuds a et b avant l'ajout du transfert. La fonction des moindres carrés à minimiser, avec l comme variable inconnue, était définie comme suit dans Makarenkov *et al.* (2006) :

$$MC(ab,l) = \sum_{dist(i,j)>l} (Min\{d(i,a) + d(j,b); d(j,a) + d(i,b)\} + l - \delta(i,j))^2 + \sum_{dist(i,j)\leq l} (d(i,j) - \delta(i,j))^2,$$

où $\delta(i,j)$ est le chemin de poids minimum entre les taxa i et j dans l'arbre de gène T_1. La fonction $MC(ab,l)$ mesure le gain en ajustement quand l'arête (a,b) de longueur l est ajoutée à l'arbre d'espèces T. Notons que l'algorithme présenté dans Makarenkov *et al.* (2006) ne tenait pas compte du pourcentage du gène transféré. C'est la nouveauté principale que nous

introduisons dans notre premier algorithme de détection de transferts partiels présenté dans la section suivante.

3.2 Premier algorithme pour la détection de transferts horizontaux de gènes partiels

Dans un arbre phylogénétique, il existe toujours un chemin unique qui permet de relier une paire de nœuds. L'ajout d'une arête de transfert horizontal crée un chemin supplémentaire entre certains nœuds. La Figure 3.2 illustre le cas où la distance d'évolution entre les taxa i et j peut être affectée par l'ajout de l'arête (b,a) représentant le transfert partiel du gène de b vers a. Nous assumons que le THG entre b et a peut affecter la distance d'évolution entre les taxa i et j si et seulement si le point de destination a est situé sur le chemin entre i et la racine de l'arbre. La distance évolutive $d_1(i,j)$ entre les taxa i et j, dans la phylogénie réticulée T (voir la Figure 3.2), peut être calculée comme suit :

$$d_1(i,j) = (1 - \alpha)\, d(i,j) + \alpha\, (d(i,a) + d(j,b)), \qquad (1)$$

où α indique la fraction du gène transféré, inconnue à l'avance, et d indique la distance entre les nœuds dans l'arbre d'espèces avant l'ajout du transfert (b,a).

La distance évolutive entre les taxa i_1 et j (voir la Figure 3.2) ne doit pas être modifiée par l'ajout de l'arête (b,a). Les autres cas que nous avons identifiés où l'ajout d'une arête ne doit pas affecter la longueur du chemin entre i et j sont illustrés dans la Figure 3.3.

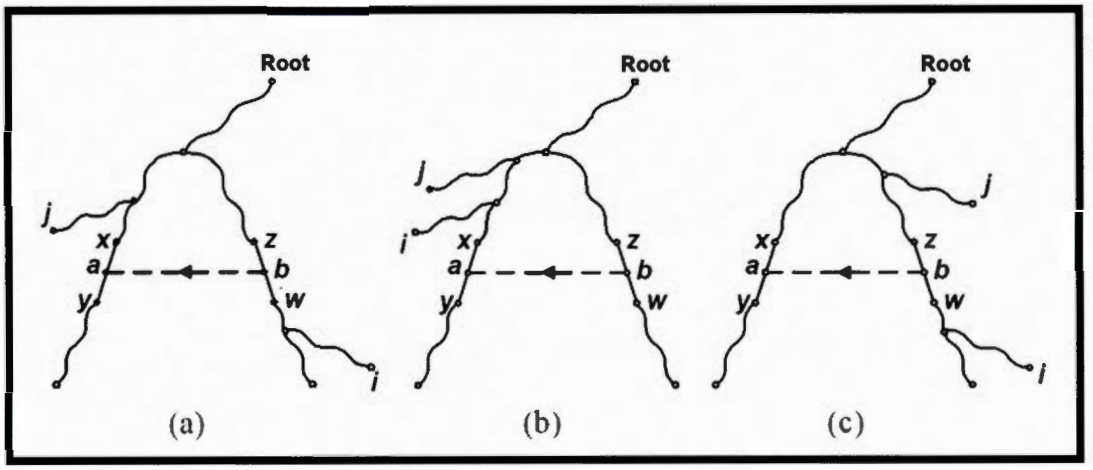


Figure 3.3 - Les cas pris en compte où la distance évolutive entre les taxa *i* et *j* ne change pas après l'ajout de la nouvelle arête (*b*,*a*) (figure tirée de Makarenkov *et al.* 2008).

La fonction des moindres carrés à minimiser, *MC*, avec un vecteur de longueurs d'arêtes $\boldsymbol{l}$ dans l'arbre d'espèces T et une fraction inconnue du gène transféré α, est définie da la manière suivante :

$$MC(L,\alpha) = \sum_{ij \in S}((1-\alpha) \sum_{k \in path(ij)} l_{ij}^{k} + \alpha(\sum_{k \in path(ia)} l_{ia}^{k} + \sum_{k \in path(jb)} l_{jb}^{k}) - \delta(i,j))^2 + \sum_{ij \notin S}(\sum_{k \in path(ij)} l_{ij}^{k} - \delta(i,j))^2 \rightarrow min, \quad (2)$$

où $\delta(i,j)$ est la valeur de la dissimilarité initiale (distance évolutive donnée) entre i et j, l_{ij}^{k} est la longueur de l'arête k du chemin (ij) dans T, α est la fraction du gène transférée ($0 \leq \alpha \leq 1$) et S est l'ensemble des paires de taxa $\{ij\}$, tels que le transfert (b,a) peut affecter la distance évolutive entre eux.

Afin de démontrer que l'optimisation du transfert partiel de gène par les moindres carrés est NP-difficile, le problème suivant peut être formulé :

Données : *L'arbre phylogénétique d'espèces T (avec la matrice de distances d associée à T sur l'ensemble de taxa X), la dissimilarité de gène δ sur X et une valeur fixe non-négative ε.*

Problème : *Trouver le nombre minimal de transferts partiels k tels que :*

$$MC = \sum_i \sum_j (d_k(i,j) - \delta(i,j))^2 \leq \varepsilon\,, \tag{3}$$

où $d_k(i,j)$ est la distance entre les taxa i et j, calculée en utilisant les Formules 1 et 2 dans le réseau phylogénétique T_k obtenu à partir de T après l'ajout de k transferts de gènes partiels.

Théorème 1. *Le problème du nombre minimal de transferts partiels de gène (MNPGT) est NP-difficile.*

La preuve de ce théorème est basée sur une réduction en un temps polynomial du *problème de transferts des sous-arbres* (le problème SPR) qui consiste à trouver le nombre minimal de transferts complets de gène nécessaire pour transformer un arbre d'espèces T donné en un arbre de gène T' donné. Pour la démonstration de la NP-complétude du problème SPR, le lecteur est référé à l'article de Hein *et al.* (1996). Le problème SPR est identique à celui de l'ajout à T du nombre minimal de transferts complets de gène tels que :

$$MC = \sum_i \sum_j (d_k(i,j) - \delta(i,j))^2 \leq 0\,;$$

(*i.e.*, le cas où $\varepsilon = 0$ est considéré), où $d_k(i,j)$ est la distance entre i et j dans l'arbre phylogénétique (*i.e.*, un cas particulier d'un réseau phylogénétique). Ici, l'arbre T_k serait obtenu à partir de T par l'ajout de k transferts complets (*i.e.*, un cas particulier d'un transfert partiel) et $\delta(i,j)$ serait la matrice de distances associée à l'arbre de gène T'.

Il est important de mentionner que plusieurs contraintes temporelles importantes doivent être incorporées dans le modèle du transfert partiel, en plus de celles qui sont déjà prises en compte dans le modèle du transfert complet (voir la Figure 2.4), pour identifier les interactions entre les THGs partiels qui ne sont pas éligibles du point de vue biologique.

Mentionnons que la règle interdisant les transferts croisés présentée sur la Figure 2.4b est automatiquement prise en compte dans les modèles de transferts de gènes complets et partiels, où ces violations seraient équivalentes à la violation de la contrainte de la même

lignée (voir la Figure 2.4 ou Page et Charleston, 1997 et 1998). Nous avons aussi identifié deux cas, où la distance évolutive entre les taxa *i* et *j* peut être affectée par des transferts multiples (voir la Figure 3.4a et b); et, deux cas, où ces distances ne doivent pas être affectées par ces transferts multiples (Figure 3.4c et d). Le fait de ne pas prendre en compte ces contraintes, peut conduire à des transferts mutuellement incompatibles.

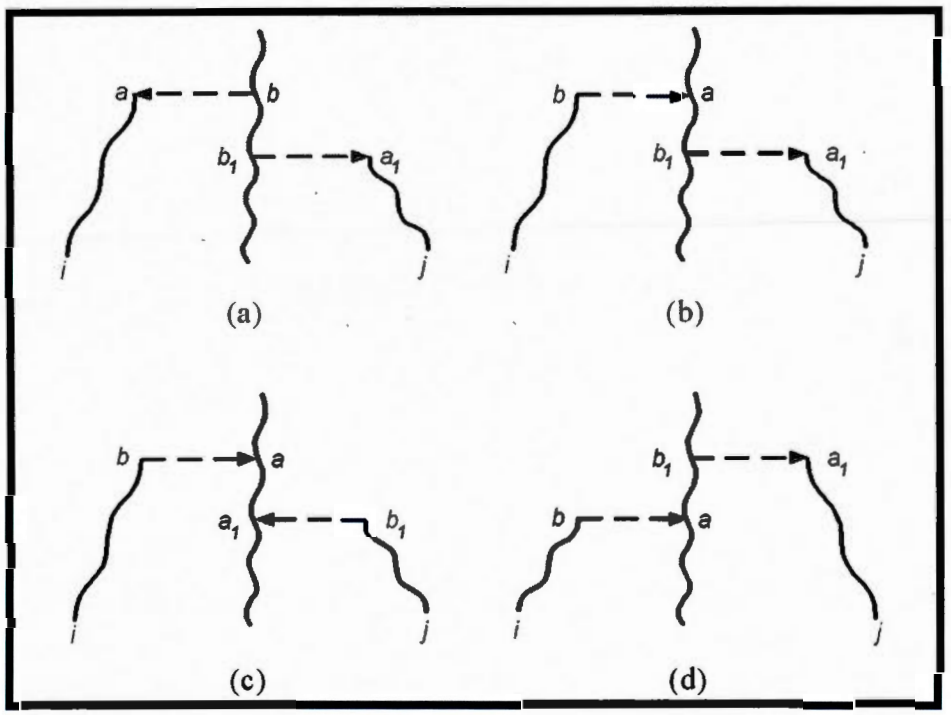


Figure 3.4 - Distance entre les feuilles *i* et *j* doit prendre en compte les deux transferts présentés dans les portions (a) et (b) de la figure. Cette distance ne doit pas dépendre des deux transferts dans les cas présentés dans les sections (c) et (d) de la figure (tirée de Makarenkov *et al.*, 2008).

Supposons qu'un transfert partiel entre les arêtes (z,w) et (x,y) (*i.e.*, de b à a dans la Figure 3.2) de l'arbre d'espèces T a eu lieu. Les longueurs de toutes les arêtes dans T sont réévaluées selon les moindres carrés après l'ajout de l'arête (b,a), alors que la longueur de (b,a) est supposée être 0.

Lors du processus de réévaluation des longueurs des arêtes de T, nous avons tout d'abord imposé une condition sur la valeur du paramètre α (Équation 1) qui représente la fraction transférée du gène. Ce paramètre peut être estimé de deux manières : soit en comparant les séquences correspondantes aux sous-arbres enracinés par les nœuds y et w, soit par le test de différentes valeurs de α dans le problème d'optimisation.

Si la valeur du paramètre α est fixée, alors le système d'équations établissant la correspondance entre les distances génétiques expérimentales et les distances dans le réseau de transferts se réduit en un système linéaire. Ce système, qui possède généralement un nombre plus élevé de variables (*i.e.*, longueurs des arêtes de T) que d'équations (*i.e.*, paires de distances dans T; le nombre d'équations est toujours $n(n\text{-}1)/2$ pour n taxa), peut être résolu en utilisant l'approximation par les moindres carrés.

Maintenant, examinons la résolution de ce problème d'approximation. Soit $\mathbf{A}_\alpha$ la matrice de dimension $n(n\text{-}1)/2 \times m$, chaque ligne correspondant à une paire de taxa dans X, où n est le nombre de taxa dans X et m est le nombre d'arêtes dans T. La valeur $a_{ij,e}$ de cette matrice, correspondant à la paire de taxa ij et à l'arête e, est égale soit à 1, soit à α, soit à $1\text{-}\alpha$ (si l'arête e se trouve sur le chemin (ij) dans T), ou est égale à 0 sinon. Soit ℓ le vecteur de longueurs d'arêtes à m éléments et $\mathbf{d}$ le vecteur de distances de gène à $n(n\text{-}1)/2$ éléments.

En fixant la valeur de α (*e.g.*, les valeurs 0, 0.1, 0.2, ..., et 1.0 ont été testées dans notre étude), nous obtenons un système d'équations linéaires avec $n(n\text{-}1)/2$ équations et m inconnues : $\mathbf{A}_\alpha \times \ell = \mathbf{d}$. Quand $n \geq 4$, ce système a plus d'équations que d'inconnues. Il peut donc être résolu par approximation de la façon suivante :

$$(\mathbf{A}_\alpha \times \ell - \mathbf{d})^2 \rightarrow \min. \tag{4}$$

Après avoir pris le gradient, nous obtenons:

$$\mathbf{A}_\alpha^t \times (\mathbf{A}_\alpha \times \ell - \mathbf{d}) = 0. \tag{5}$$

À la suite de manipulations algébriques, nous avons :

$$\mathbf{A}_\alpha^t \times \mathbf{A}_\alpha \times \ell = \mathbf{A}_\alpha^t \times \mathbf{d}. \tag{6}$$

Donc, nous obtenons : $\mathbf{B} \times \ell = \mathbf{c}$, où **B** est une matrice ($m \times m$) et **c** est un vecteur à m composantes.

En se basant sur Barthélemy et Guénoche (1991) et Makarenkov et Leclerc (1999), nous pouvons appliquer la méthode de Gauss-Seidel légèrement modifiée pour résoudre le système ci-dessus. La méthode consiste à décomposer **B** en sa diagonale (Δ), sa composante triangulaire supérieure stricte (**-F**) et sa composante triangulaire inférieure stricte (**-E**):

$$\mathbf{B} = \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1m} \\ b_{21} & b_{22} & \cdots & b_{2m} \\ \cdots & \cdots & \cdots & \cdots \\ b_{m1} & b_{m2} & \cdots & b_{mm} \end{pmatrix} = \begin{pmatrix} & & -\mathbf{F} \\ & \Delta & \\ -\mathbf{E} & & \end{pmatrix} = \Delta - \mathbf{E} - \mathbf{F}. \quad (7)$$

Nous appliquons maintenant la procédure itérative suivante :

$$\Delta \times \ell^{(k+1)} = \mathbf{E} \times \ell^{(k+1)} + \mathbf{F} \times \ell^{(k)} + \mathbf{c}, \quad (8)$$

qui nous permet de calculer progressivement les composantes du vecteur $\ell(j)^{(k+1)}$, correspondant aux longueurs des arêtes de T à la k+1-ème itération, à partir de $\ell(j)^{k}$. Si la valeur calculée de $\ell(j)^{(k+1)}$ est négative, elle est remplacée par la valeur de 0. Cette opération est équivalente à la projection sur le cône $\ell \geq 0$, qui assure une solution appropriée (*i.e.*, non-négative).

L'équation itérative exacte utilisée dans cette méthode est la suivante, pour tous $j = 1, 2, \ldots, m$:

$$\ell(j)^{(k+1)} = (- (\sum_{j+1 \leq i \leq m} b_{ij} \ell(j)^{(k)}) - (\sum_{1 \leq i \leq j-1} b_{ij} \ell(j)^{(k+1)}) + c_j) / b_{jj}. \quad (9)$$

Les principales étapes de notre premier algorithme pour la détection de transferts de gènes partiels peuvent donc être énoncées comme suit :

Étape préliminaire. Cette étape correspond à l'étape préliminaire discutée dans le contexte du modèle du transfert complet. Elle consiste à l'inférence des phylogénies d'espèces et de gène, notées respectivement T et T', dont les feuilles sont étiquetées par le même ensemble X de n taxa. Nous utilisons le critère des moindres carrés comme l'unique critère

d'optimisation quand nous modélisons les transferts horizontaux partiels (car les trois autres critères considérés dans le chapitre II, *RF*, *QD* et *BD*, s'appliquent uniquement à des topologies d'arbres).

***Étape* 1..*k*.** Nous testons toutes les connexions possibles entre les paires d'arêtes dans l'arbre *T*. Pour chaque THG satisfaisant les contraintes d'évolution, nous effectuons les opérations suivantes :

- Nous fixons la valeur de la fraction transférée du gène α et calculons les longueurs optimales ℓ des arêtes dans l'arbre d'espèces T (ou dans le réseau, commençant par l'étape 2) en utilisant la méthode itérative de Gauss-Seidel (Formules 8 et 9).
- Nous retournons au système d'équations original : $\mathbf{A}_a \times \ell = \mathbf{d}$. Nous fixons les valeurs du vecteur ℓ trouvées en utilisant la méthode de Gauss-Seidel et résolvons le problème par les moindres carrés en considérant comme inconnu le paramètre α.
- Nous fixons la valeur optimale de α trouvée et répétons le calcul jusqu'à ce que les deux paramètres inconnus convergent vers une solution.

Toutes les paires d'arêtes éligibles (*i.e.*, satisfaisant les contraintes biologiques) dans T sont traitées de la même façon. Le THG qui produit la plus petite valeur du coefficient des moindres carrés, *MC*, et qui satisfait les contraintes biologiques définies, sera sélectionné pour l'ajout dans l'arbre d'espèces T, le transformant ainsi en un réseau phylogénétique. L'algorithme est exécuté jusqu'à ce qu'un nombre fixe de transferts partiels, τ, soit trouvé et ajouté à T ou que la valeur de *MC* soit plus petite que le seuil préétabli. La complexité algorithmique de cette méthode est $O(\tau n^5)$ pour ajouter τ transferts horizontaux de gène partiels dans un arbre phylogénétique d'espèces à n feuilles. Les simulations que nous avons réalisées avec cet algorithme ont montré qu'il est plus performant que la méthode décrite dans Makarenkov *et al.* (2006), mais généralement moins performant que le deuxième algorithme de détection de transferts horizontaux de gènes que nous présentons dans la section suivante (voir la Figure 3.6 pour plus de détails).

Le schéma algorithmique de la procédure de détection de THGs partiels que nous avons décrite ci-dessus, se présente comme suit (voir la Figure 3.5) :

DÉBUT

Inférer les arbres d'espèces T et de gène T' sur le même ensemble d'espèces (i.e., feuilles);

Enraciner les arbres T et T' selon des évidences biologiques existantes ou en utilisant un *outgroup* ou un *point médian*;

Le critère d'optimisation retenu est le critère des moindres carrés *MC*;

Calculer la valeur initiale du critère *MC* entre T et T';

$T_0 = T$;

$k = 0$; //k est l'indice de l'étape

Tant que (*MC* > *Seuil* et $k \leq \tau$) {

Trouver l'ensemble des *paire d'arêtes éligibles* à l'étape k, qui est représenté par E_AE_k;

L'ensemble E_AE_k contient seulement les transferts satisfaisant aux contraintes d'évolution;

Fixer la valeur de la fraction transférée du gène α et calculer les longueurs optimales ℓ des arêtes dans l'arbre d'espèces (ou dans le réseau, commençant par l'étape 2) T en utilisant la méthode itérative de Gauss-Seidel (voir les Formules 8 et 9);

Retourner au système d'équations original : $\mathbf{A}_a \times \ell = \mathbf{d}$ et fixer les valeurs du vecteur ℓ trouvées en utilisant la méthode de Gauss-Siedel. Résoudre le problème par les moindres carrés en considérant comme inconnu le paramètre α;

Répéter les calculs ci-haut jusqu'à ce que les deux paramètres inconnus α et ℓ convergent;

Recalculer la valeur du critère *MC* après l'ajout du transfert optimal sélectionné;

$k = k + 1$;

}

FIN

Figure 3.5 - Premier algorithme de détection de THGs partiels.

3.3 Deuxième algorithme pour la détection de transferts horizontaux de gènes partiels

Dans cette section nous décrivons notre deuxième algorithme d'inférence de THGs partiels (Boc *et al.*, 2011). Les principales étapes de cet algorithme, qui cherche à produire un scénario optimal de transferts partiels d'un gène donné pour un groupe d'espèces considéré, sont présentées dans la Figure 3.6. Une validation par bootstrap est réalisée pour chaque transfert partiel hypothétique détecté lors de l'exécution de l'algorithme. Nous incluons dans la solution finale uniquement les transferts avec les valeurs de bootstrap significatives. Une procédure de fenêtre coulissante a été mise au point pour tester différents fragments de l'alignement de séquences multiples (ASM). Remarquons que quelques approches basées sur une fenêtre coulissante ont été développées pour détecter les recombinaisons (Ray, 1998; Paraskevis *et al.*, 2005), mais aucun de ces travaux ne traite du problème de transfert de gènes partiels. Les principales étapes de notre algorithme sont les suivantes :

Étape préliminaire. Soit X un ensemble d'espèces étudiées, l'ASM un alignement de séquences multiples de taille l, et $S_{i,j}$ le fragment de l'ASM, étant analysé, et situé entre les sites i et j, où $1 <= i < j <= l$. Définissons aussi la taille de la fenêtre coulissante w ($w = j - i + 1$) et la taille du pas de progression s. Inférons l'arbre phylogénétique d'espèces T. D'habitude, un arbre basé sur les caractères morphologiques ou sur une molécule qui est supposée être réfractaire aux transferts horizontaux de gènes (*e.g.*, 16S sRNA ou 23S sRNA) joue le rôle de l'arbre d'espèces. L'arbre T doit être enraciné en respectant les hypothèses d'évolution connues. Si aucune connaissance permettant d'enraciner les arbres n'est disponible, des stratégies d'enracinement par *outgroup* (en utilisant un groupe des espèces extérieur au groupe étudié) ou par *midpoint* (en utilisant l'arête médiane de l'arbre) peuvent être considérées. L'enracinement correct des arbres est primordial car la racine permet de prendre en compte les règles d'évolution (voir la Figure 2.4) qui doivent être respectées lorsqu'on infère des transferts horizontaux. Fixons la taille de la fenêtre coulissante w et la taille du pas s (dans nos expériences, les tailles de fenêtres égales à $l/5$, $l/4$, $l/3$, $l/2$ et un pas de progression de 10 nucléotides ont été utilisés).

***Étape* 1..*k*.** Fixons la position de la fenêtre coulissante dans l'intervalle $[i,j]$, où $i = 1 + s(k - 1)$ et $j = i + w - 1$. Si $i + w - 1 > l$, alors $j = l$. Inférons un arbre de gène partiel T' caractérisant l'évolution du fragment de l'ASM localisé dans l'intervalle $[i, j]$. Dans cette étude, la méthode PhyML (Guindon et Gascuel, 2003) a été employée pour inférer les arbres de gène partiels. Appliquons un algorithme de détection existant pour inférer un scénario de THGs partiels associé à l'intervalle $[i, j]$. Ici, nous avons utilisé l'algorithme *HGT-Detection* (Boc *et al.*, 2010) pour inférer des transferts complets, mais n'importe quel autre algorithme pourrait être utilisé à sa place. Cet algorithme de détection de THGs est plus rapide et dans la plupart des cas aussi plus précis que les populaires algorithmes *LatTrans* (Hallett et Lagergren, 2001) et *RIATA-HGT* (Nakhleh *et al.*, 2005). La dissimilarité de bipartitions présentée dans le chapitre II a été utilisée comme critère d'optimisation. Une procédure servant à évaluer la fiabilité des transferts partiels obtenus (*i.e.*, support de bootstrap) a aussi été développée (r réplicats de l'arbre de gène partiel ont été considérés lors de la validation par bootstrap). Cette procédure, basée sur le principe du bootstrap, prend en compte l'incertitude des arbres de gène partiels ainsi que le nombre de fois qu'un transfert donné apparait dans tous les scénarios de coût minimum. Parmi les THGs partiels obtenus, nous avons retenu seulement ceux ayant un score de bootstrap significatif (par rapport à un seuil défini).

Étape finale. Établissons une liste des THGs partiels prédits. Identifions les intervalles entrelacés donnant lieu à des transferts partiels identiques (*i.e.*, les mêmes donneurs et receveurs et la même direction). Ré-exécutons l'algorithme de détection de THGs pour tous les intervalles entrelacés, en considérant leur longueur totale dans chaque cas, produisant les THGs partiels identiques. Si ces THGs partiels sont trouvés à nouveau, c'était habituellement le cas dans nos simulations, pour le fragment de séquences situé dans les intervalles entrelacés, évaluons leur support de bootstrap et, dépendamment du support obtenu, incluons-les ou non dans la solution finale.

La complexité de notre algorithme est comme suit :

$$O(r \times (\frac{(l-w)}{s} \times (C(PhIn) + \tau \times n^4))), \qquad (10)$$

où w est la taille de la fenêtre coulissante, l est la longueur de l'alignement de séquences multiples considéré, s est le pas de progression, $C(PhIn)$ est la complexité de la méthode d'inférence d'arbres phylogénétiques (*e.g.*, PhyML) utilisée pour inférer les phylogénies à partir des fragments de séquences dans la fenêtre coulissante, r est le nombre de réplicats dans le bootstrap (r était égal à 100 dans nos simulations), n est le nombre d'espèces et τ est le nombre moyen de transferts inférés pour un fragment de séquences de taille w. Cette complexité algorithmique s'explique par le fait que chacun des r réplicats de la méthode est appliqué à chacune des $(l - w)/s$ positions fixes de la fenêtre coulissante et que pour chacune de ces positions fixes on exécute une méthode d'inférence d'arbres phylogénétiques *PhIn* et l'algorithme de détection de THGs complets, *HGT-Detection*, dont la complexité est $O(\tau n^4)$.

Le schéma algorithmique complet du deuxième algorithme de détection de THGs partiels se présente comme suit (voir la Figure 3.6) :

DÉBUT

Soit : X un ensemble d'espèces étudiées;

ASM un alignement de séquences multiples de taille l;

$S_{i,j}$ le fragment de l'ASM étant analysé et situé entre les sites i et j, où $1 \leq i < j \leq l$;

w la taille de la fenêtre coulissante ($w = j - i + 1$);

s le pas de progression;

r le nombre de réplicats dans le bootstrap;

Inférer l'arbre d'espèces T;

Enraciner l'arbre T selon des évidences biologiques existantes ou en utilisant un *outgroup* ou un *point médian*;

Fixer la taille de la fenêtre coulissante w et du pas s.

Pour ($k = 1$; $k < l - w$; $k = k + s$) {

$i = 1 + s\,(k - 1)$;

$j = i + w - 1$;

Inférer avec *PhyML* (Guindon et Gascuel, 2003) un arbre de gène partiel T'_{ij} caractérisant l'évolution du fragment de l'ASM localisé dans l'intervalle $[i, j]$;

Appliquer un algorithme de détection de THGs complets pour inférer un scénario de THGs partiels associés à l'intervalle $[i, j]$. L'algorithme *HGT-Detection* (Boc *et al.*, 2010) a été utilisé pour inférer des transferts complets;

Exécuter la procédure pour évaluer la fiabilité des transferts partiels obtenus en générant r réplicats de l'arbre de gène partiel T'_{ij}. Cette procédure, basée sur le principe de bootstrap, prend en compte l'incertitude des arbres de gène partiels, ainsi que le nombre de fois qu'un transfert donné apparait dans tous les scénarios de coût minimum (*i.e.*, comportant le nombre minimum de THGs nécessaires pour réconcilier les arbres T et T'_{ij});

}

FIN

Figure 3.6 - Deuxième algorithme de détection de THGs partiels.

3.4 Comparaison des deux algorithmes de détection de transferts horizontaux de gènes partiels

Dans cette section nous effectuons une comparaison des deux algorithmes de détection de transferts horizontaux de gènes partiels décrits ci-haut. Des simulations Monte Carlo ont été effectuées pour comparer l'efficacité des deux algorithmes de détection de THGs partiels en termes de faux positifs et de faux négatifs. Le comportement des deux algorithmes proposés a été examiné en fonction du nombre d'espèces observées, qui était respectivement 8, 16, 32 et 64 (voir la Figure 3.7). La procédure de simulations incluait les quatre étapes suivantes :

Première étape. Des arbres binaires d'espèces avec 8, 16, 32 et 64 feuilles ont été créés en utilisant la procédure de génération d'arbres aléatoires de Kuhner et Felsenstein (1994); voir le chapitre II pour plus de détails sur cette procédure. Les longueurs des arêtes des arbres ont été générées en utilisant une distribution exponentielle. En suivant l'approche de Guindon et Gascuel (2002), nous avons ajouté du bruit aux arêtes afin de modéliser un écart par rapport à l'hypothèse de l'horloge moléculaire. Les arbres aléatoires produits par cette procédure avaient une profondeur de $O(\log(n))$, où n était le nombre d'espèces (*i.e.*, le nombre de feuilles dans un arbre phylogénétique binaire).

Deuxième étape. Nous avons exécuté le programme *SeqGen* (Rambaut et Grassly, 1997) pour générer des alignements de séquences multiples de protéines le long des arêtes des arbres d'espèces construits à la première étape. Le programme *SeqGen* a été utilisé avec le modèle de substitution de protéines JTT (Jones *et al.*, 1992), une distribution Gamma estimée, un nombre de catégories de taux de substitution égal à 4 et une proportion des sites invariables égale à 0. Des séquences de protéines avec 500 caractères (*i.e.*, acides aminés) ont été générées.

Troisième étape. Pour chaque arbre d'espèces T, nous avons par la suite généré des arbres de gène, ayant le même nombre de feuilles, en effectuant des déplacements ou opérations SPR aléatoires des sous-arbres de T. Un modèle satisfaisant toutes les contraintes d'évolution plausibles a été implémenté pour générer les THGs partiels aléatoires. Pour chaque arbre d'espèces, 1 à 5 déplacements SPR aléatoires ont été effectués et différents arbres de gènes

T', englobant entre 1 et 5 THGs partiels, ont été générés. Pour chaque arbre de gène, les fragments de séquences dans les sous-arbres affectés par les THGs ont été régénérés avec le programme *SeqGen*. Nous avons fixé la taille de chaque séquence transférée à 200 acides aminés. Les alignements de séquences multiples obtenus contenaient donc des blocs de séquences affectés par des THGs.

Quatrième étape. Nous avons exécuté les deux algorithmes pour chaque arbre d'espèces généré et l'alignement de séquences multiples associé qui était affecté par les THGs partiels. Dans le cas du deuxième algorithme, la taille de la fenêtre coulissante a été progressivement fixée à 100, 200, 300, 400, puis 500 acides aminés; 100 réplicats de chaque arbre de gène partiel *T'* ont été générés pour évaluer le support de bootstrap des arêtes de *T'* dans un premier temps, puis le support des THGs partiels obtenus dans un deuxième temps. Les arbres qui avaient le support de bootstrap moyen inférieur à 60% ont été retirés de l'analyse. Parmi les THGs obtenus, seuls les transferts avec un bootstrap supérieur à 90% ont été retenus dans la solution finale (toujours dans le cas du deuxième algorithme). Finalement, nous avons estimé le taux de détection (*i.e.*, les vrais positifs seulement) et le taux de faux positifs pour les deux algorithmes en fonction du nombre d'espèces et de transferts générés.

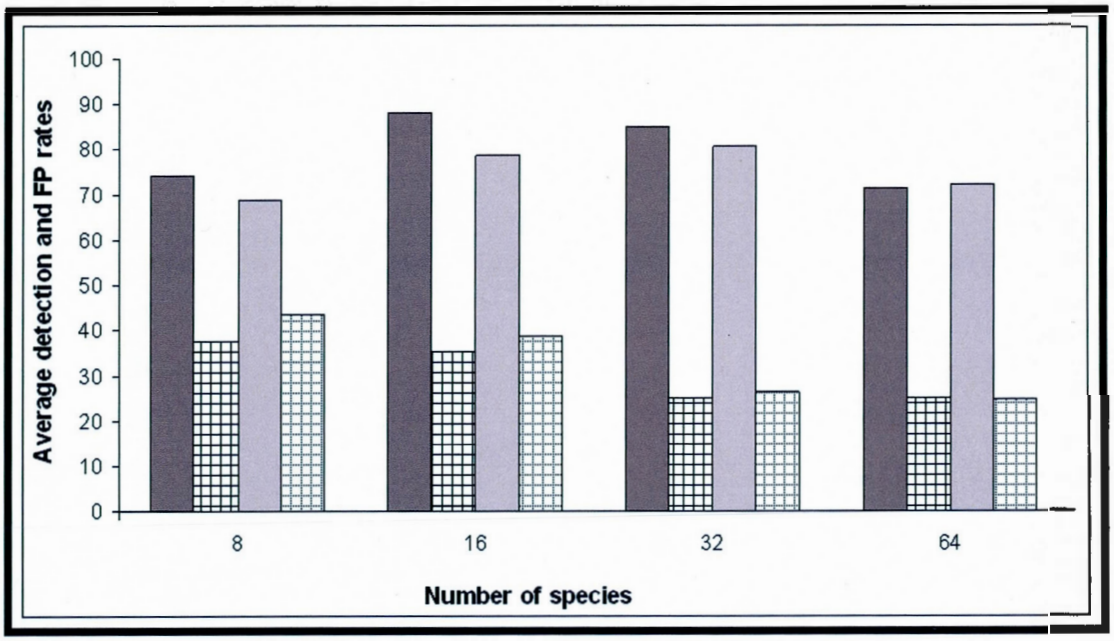


Figure 3.7 - Taux moyens de détection (vrais et faux positifs) pour les cas de 1 à 5 transferts partiels générés, obtenus par les deux algorithmes de détection de THGs partiels présentés dans les sections 3.2 et 3.3 de la thèse. La première colonne (en gris foncé) représente le taux de détection du deuxième algorithme, la deuxième colonne (en carrés foncés) représente le taux de faux positifs du deuxième algorithme, la troisième colonne (en gris clair) représente le taux de détection du premier algorithme et la quatrième colonne (en carrés clairs) représente le taux de faux positifs du premier algorithme.

La Figure 3.7 met en lumière les différences entre les taux de détection moyens et les taux de faux positifs moyens fournis par les deux algorithmes de détection de THGs partiels, en fonction du nombre d'espèces. Les moyennes ici ont été calculées à partir des résultats obtenus pour les cas de 1 à 5 THGs générés. On peut constater que le deuxième algorithme a généralement fourni de meilleurs résultats que le premier. Le premier algorithme a été légèrement supérieur au deuxième seulement pour le cas des 64 espèces, alors que pour les trois autres cas (8, 16 et 32 espèces), le deuxième algorithme avait un meilleur taux de détection et un plus faible nombre de faux positifs. Notons que le taux de détection moyen obtenu par le deuxième algorithme était toujours supérieur à 70% (79,6% en moyenne) et que son taux moyen de faux positifs était toujours inférieur à 40% (30,8% en moyenne). C'est

pour cette raison que nous allons l'examiner d'une façon plus détaillée, en effectuant le profilage du programme qui l'implémente.

D'autres simulations détaillées ont été effectuées pour tester l'efficacité du deuxième algorithme de détection de transferts partiels. Nous avons examiné comment cet algorithme se comporte en fonction du nombre d'espèces observées (*i.e.*, la taille de l'arbre) et du nombre de transferts partiels générés. Pour chaque ensemble de paramètres (taille de l'arbre, nombre de THGs générés), 100 jeux de données répliqués ont été testés. Nous avons obtenus les meilleurs taux de détection pour les arbres de 16 espèces. Les résultats variaient de 100%, pour un transfert, à 82%, pour cinq transferts. On peut noter que le taux de faux positifs pour les arbres de 16 espèces était habituellement inférieur à 40%. Les mêmes tendances ont été observées pour les autres tailles d'arbres. Selon des tests additionnels, les performances du deuxième algorithme peuvent encore être améliorées en ajustant les paramètres des simulations dépendamment de la nature des données étudiées, en particulier, en ajustant le seuil de bootstrap moyen des arbres de gènes. Les résultats des simulations suggèrent que le deuxième algorithme de détection de transferts partiels peut être très utile pour identifier des gènes mosaïques. Notons que les plus faibles taux de faux positifs ont été obtenus pour les arbres de 32 et 64 feuilles. Alors qu'en moyenne les taux de détection de THGs partiels étaient légèrement plus faibles que ceux obtenus par les algorithmes *LatTrans* (Hallett et Lagergren, 2001) et *HGT-Detection* (Boc *et al.*, 2010) pour les transferts complets, il est important de noter que le problème de détection de THGs partiels est beaucoup plus complexe que celui de détection de THGs complets, principalement en raison de la forte similarité des fragments de séquences situés dans les blocs d'alignements multiples affectés par des THGs et à cause des situations où ces blocs se chevauchent et cachent de réels transferts de gènes. Mentionnons que les performances de l'algorithme s'amélioraient quand la longueur des blocs de séquences transférées augmentait.

3.5 Profilage du deuxième algorithme de détection de transferts horizontaux de gènes partiels

Contrairement au programme de détection de THGs complets qui a été implémenté en langage *C++*, le deuxième algorithme de détection de THGs partiels qui s'exécute à travers

un script développé en langage *Perl* (Wall, 1994), qui invoque entre autres les fonctions principales suivantes : *Robinson_and_Foulds* - pour le calcul de la distance topologique de *Robinson et Foulds*, *PhyML* - pour l'inférence d'arbres et *HGT-Detection* - pour inférer un scénario des THGs pour les différents intervalles considérés (voir la Figure 3.6 pour le schéma algorithmique complet du deuxième algorithme de détection de THGs partiels). Nous avons utilisé la commande *dprofpp* de *Linux* pour effectuer le profilage de notre script *Perl*. La commande *dprofpp* interprète les données produites par un profileur, tel que celui que nous avons installé et employé : *Devel::DProf*. Lors de notre profilage, nous avons utilisé les paramètres d'entrée de Delwiche et Palmer (1996) comportant 42 espèces et un alignement de séquences multiples de taille 532 acides aminés (532 sites). Notons que la fonction *main::executeLinuxCommandInLoop* inclut toutes les commandes de gestion *Linux* appelées dans la boucle principale du programme (voir la boucle *Pour* de la Figure 3.6), tandis que la fonction *main::executeLinuxCommandOutLoop* inclut toutes les commandes de gestion *Linux* appelées en dehors de la boucle principale (voir l'Annexe C.1 pour plus de détails sur le script *Perl* développé).

Les résultats du profilage de notre script *Perl* sont les suivants :

% de Temps	Temps d'exécution (en sec)	Temps cumulatif (en sec)	Nombre d'appels	Temps par appel (en sec)	Temps moyen par appel (en sec)	Nom de la fonction
37.5	**0.029**	**0.029**	**100**	**0.0003**	**0.0003**	**main::executeLinuxCommandInLoop**
26.1	**0.020**	**0.020**	**10**	**0.0020**	**0.0020**	**main::executePhyML**
13.0	**0.010**	**0.010**	**10**	**0.0010**	**0.0010**	**main::Robinson_and_Foulds**
11.7	0.009	0.009	14	0.0006	0.0006	main::executeLinuxCommandOutLoop
11.7	**0.009**	**0.009**	**10**	**0.0009**	**0.0009**	**main::executeHGT_Detection**
0.00	0.000	0.000	51	0.0000	0.0000	main::file_get_contents
0.00	0.000	0.000	50	0.0000	0.0000	main::file_put_contents
0.00	0.000	0.000	1	0.0000	0.0000	main::clean
0.00	0.000	0.000	1	0.0000	0.0000	main::chargerSequence

*sec : seconde

Tableau 3.1 - Résultats du profilage du deuxième algorithme de détection de THGs partiels.

Les différentes informations fournies par le Tableau 3.1 sont les suivantes :

- Le pourcentage du temps (% du temps) : représente le pourcentage du temps passé dans la fonction.
- Le temps d'exécution (en secondes) : représente le temps moyen passé dans la fonction (ce temps n'inclut pas le temps passé dans ses fonctions descendantes).
- Le temps cumulatif (en secondes) : représente le temps total passé dans la fonction ainsi que dans ses fonctions descendantes.
- Le nombre d'appels : représente le nombre de fois que la fonction a été appelée.

- Le temps par appel (en secondes) : représente le temps moyen passé dans la fonction.
- Le temps moyen par appel (en secondes) : représente le temps moyen par appel passé dans la fonction ainsi que dans ses fonctions descendantes.
- Le nom : représente le nom de la fonction.

En se basant sur les résultats présentés dans le Tableau 3.1, nous pouvons conclure que les fonctions les plus intéressantes du point de vue de l'optimisation sont les suivantes : *main::Robinson_and_Foulds*, *main::executePhyML* et *main::executeHGT_Detection*. On peut aussi remarquer, en observant la Figure 3.6, que les trois fonctions mentionnées sont appelées dans la boucle principale du programme - voir la ligne :

$$Pour\ (k = 1;\ k < l - w;\ k = k + s)$$

dans le schéma algorithmique du deuxième algorithme de détection de THGs partiels.

Une optimisation de ce programme pourrait donc se faire des deux manières suivantes :

(1) L'exécution en parallèle de la boucle *Pour* : chaque position fixe de la fenêtre coulissante considérée serait traitée indépendamment des autres, où le nombre de processus indépendants pourrait être déterminé par l'expression suivante :

$(l - w)/s$.

(2) La parallélisation des algorithmes *PhyML* (voir Guindon *et al.*, 2010 concernant la parallélisation de cet algorithme), *HGT-Detection* (voir le chapitre IV, section 4.7 concernant la parallélisation de cet algorithme) et *Robinson et Foulds* (Robinson et Foulds, 1981).

Dans le cadre de ce projet doctoral, nous n'avions pas la tâche de paralléliser le deuxième algorithme de détection de THGs partiels. Une parallélisation de cet algorithme pourrait être effectuée en se basant sur les résultats du profilage et sur la discussion présentés ci-dessus.

3.6 Résumé

Dans ce chapitre, nous avons présenté deux nouveaux algorithmes pour la détection de transferts horizontaux de gènes partiels. Deux articles traitant de l'inférence des transferts horizontaux de gènes partiels, dont je suis co-auteur (Makarenkov *et al.*, 2008; Boc *et al.*, 2011) ont été rédigés et publiés dans le cadre de ce projet doctoral.

Ma contribution dans le développement de ces deux algorithmes était la suivante :

- Élaboration des deux algorithmes présentés, y compris la formulation et la preuve de NP-complétude du problème de recherche des transferts horizontaux partiels en utilisant le modèle des moindres carrés (les deux dernières taches ont été effectuées en collaboration avec A. Boc).
- Développement d'une partie importante du code source en *C/C++* et des scripts *Perl* pour ces algorithmes (voir l'Annexe C).
- Élaboration des tests servant à assurer la validité (tests de bootstrap) et la qualité des résultats du deuxième algorithme de détection de THGs.
- Implémentation des simulations permettant de comparer les résultats des deux algorithmes de détection de THGs partiels (voir la Figure 3.7).
- Développement de l'interface Web (voir le chapitre V) pour le deuxième algorithme de détection de THGs partiels.

CHAPITRE IV

PARALLÉLISATION DE L'ALGORITHME DE DÉTECTION DE TRANSFERTS HORIZONTAUX DE GÈNES COMPLETS

Dans ce chapitre, nous introduisons, tout d'abord, les définitions et les notions de base de la programmation parallèle. Afin de proposer une solution efficace au problème de la parallélisation d'algorithmes bioinformatiques séquentiels, nous devons premièrement comprendre l'étendue de la problématique de la programmation parallèle. Ensuite, nous présenterons brièvement un nouvel outil de parallélisation semi-automatique pour les architectures à mémoire distribuée, qui sera basé sur une série de transformations d'un programme séquentiel écrit en langage *C* (Kernighan et Ritchie, 1988) en un programme parallèle à travers la bibliothèque d'échange de messages, MPI. Cette présentation sera suivie par la description de la version parallèle du programme de détection de THGs complets, *HGT-Detection*. Finalement, nous exposerons les résultats des tests de performance du programme parallèle de détection de THGs complets, effectués sur des jeux de données de tailles différentes.

4.1 Notions de base du parallélisme

Même si la puissance des microprocesseurs continue d'augmenter, plusieurs spécialistes et observateurs affirment que la programmation parallèle est l'un des moyens les plus efficaces pour améliorer la puissance de calcul des systèmes informatiques de façon significative (Wilkinson et Allen, 2004). Au niveau des systèmes parallèles multiprocesseurs, l'optimisation du compilateur et la parallélisation sont essentielles afin de profiter de la puissance de traitement nécessaire pour exécuter des applications contenant des calculs

intensifs. La transformation d'un programme séquentiel en un programme parallèle nécessite un certain nombre de modifications du code source. Les principales étapes de cette transformation peuvent être résumées comme suit :

- La recherche des parties du programme susceptibles d'être parallélisées, ainsi que la détection des dépendances des données (Massingill *et al.*, 2000).
- La génération du code parallèle, à savoir le partage des données (Massingill *et al.*, 1999; Massingill *et al.*, 2000), l'insertion d'instructions parallèles au programme séquentiel et la synchronisation des résultats (Quinn, 2004).

Un *programme séquentiel* est un programme qui exécute une suite d'instructions guidées par un processeur (CPU) unique, c'est-à-dire une seule instruction est exécutée à la fois. Un *programme parallèle*, quant à lui, peut exécuter plusieurs instructions simultanément sur plusieurs processeurs.

En 1966, Philip Bernstein a défini les conditions exprimant la possibilité d'exécution des programmes en parallèle (Chevance, 2000) :

> « Soit P_1 et P_2 deux programmes; on suppose que chacun des programmes utilise des variables en entrée (E_1 et E_2) et produit des valeurs en sortie (S_1 et S_2). Les programmes P_1 et P_2 sont exécutables en parallèle ($P_1 \mid\mid P_2$) si, et seulement si, les conditions suivantes sont respectées : $E_1 \cap S_2 = \emptyset$, $E_2 \cap S_1 = \emptyset$, $S_1 \cap S_2 = \emptyset$.
> De façon plus générale, un ensemble de programmes P_1, P_2,.., P_n, est exécutable en parallèle si, et seulement si, les conditions de *Bernstein* sont satisfaites. »

La violation de la première condition de Bernstein ($E_1 \cap S_2 = \emptyset$) implique une dépendance des flux. C'est-à-dire le premier programme (ou tâche) produit un résultat utilisé par le second programme (voir la Figure 4.1).

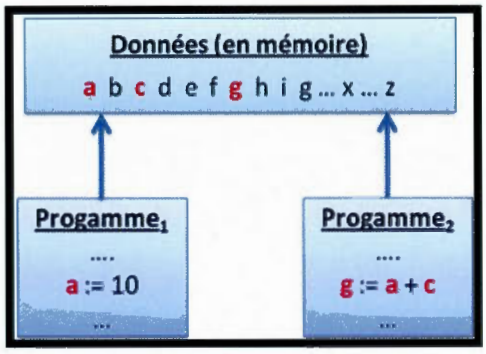


Figure 4.1 - Première condition de Bernstein.

La seconde condition de Bernstein ($E_2 \cap S_1 = \emptyset$) est une condition d'anti-dépendance. Dans ce cas, le second programme écrase (ou remplace) une variable utilisée par le premier programme (voir la Figure 4.2).

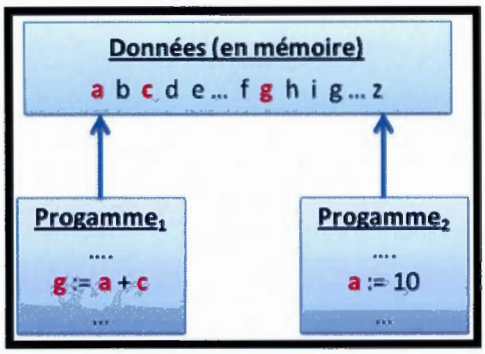


Figure 4.2 - Deuxième condition de Bernstein.

Finalement, la dernière condition de Bernstein ($S_1 \cap S_2 = \emptyset$) est une condition sur les sorties. Lorsque les deux programmes écrivent sur la même adresse en mémoire, la valeur finale est celle produite par le programme exécuté en dernier (voir la Figure 4.3).

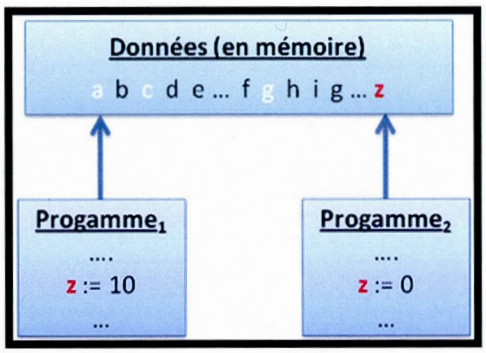


Figure 4.3 - Troisième condition de Bernstein.

La programmation parallèle a connu une croissance fulgurante au cours de la dernière décennie. Les méthodes de programmation traditionnelles, qui sont basées sur des essais successifs, ont été progressivement remplacées par de nouvelles approches basées sur des simulations et des analyses. Les ingénieurs et les scientifiques dépendent fortement des superordinateurs et des stations de travail rapides pour résoudre de nombreux problèmes importants dans des domaines tels que la météorologie, la bioinformatique, l'aérodynamique, la génétique, etc.

Il existe trois formes de parallélisme, qui sont définies en fonction de la formulation ou de la décomposition du problème (Chevance, 2000) :

- *Parallélisme de données* : « La même opération est effectuée par des processeurs différents sur des ensembles disjoints de données ».
- *Parallélisme de contrôle* ou *fonctionnel* : « Différentes opérations sont réalisées simultanément. Ce parallélisme est utilisé lorsque le programme est composé de parties indépendantes ou lorsque certaines structures de contrôle (telles que les boucles) sont susceptibles d'être exécutées en parallèle ».
- *Parallélisme de flux* : « Les opérations sur un même flux de données peuvent être enchaînées avec un certain niveau de recouvrement. C'est-à-dire que l'opération suivante peut être amorcée avant que la précédente ne soit terminée. C'est le mode de travail à la chaîne (ou pipeline) ».

La programmation parallèle consiste à concevoir, à exécuter et à agencer des programmes sur des machines parallèles afin d'en tirer le maximum en termes de performance. Elle consiste également à appliquer des modèles de programmation parallèle aux programmes séquentiels existants. Le principe général de la programmation parallèle est de diviser le problème global en tâches qui sont par la suite assignées aux différents processeurs disponibles. Au niveau des systèmes multiprocesseurs, des opérations de synchronisation des tâches sont souvent nécessaires pour assurer la cohérence du résultat final (Grama *et al.*, 2003).

Donner une classification aux architectures des systèmes informatiques est un exercice auquel de nombreux experts se sont essayés. Entre les années 1960 et le début des années 90, les ingénieurs et les scientifiques ont exploré une grande variété d'architectures parallèles. Cette exploration a connu son zénith dans les années 80 (Flynn, 1972; Hockney, 1988; Treleaven, 1988; Skillicorn, 1988; Duncan, 1990; Germain-Renaud et Sansonnet, 1991).

La classification de Flynn (1972) englobe tous les modèles de fonctionnement et toutes les architectures concevables. Cette classification est la plus couramment utilisée. Elle est assez simple et complète pour faciliter la compréhension des systèmes informatiques et sera adoptée dans le cadre de cette thèse. Elle est basée sur deux critères : le flux de données et le flux d'instructions. La classification de Flynn peut être décomposée en quatre catégories (voir la Figure 4.4) :

- L'architecture SISD (*Single Instruction on Single Data stream*, ou une instruction sur une seule donnée) : Dans cette architecture, un flux d'instructions est appliqué à un flux de données. Nous retrouvons dans cette architecture les ordinateurs traditionnels à processeur unique.
- L'architecture SIMD (*Single Instruction on Multiple Data stream*, ou une instruction sur plusieurs données) : Dans cette architecture, le même flux d'instructions est appliqué à des ensembles disjoints de données (plusieurs flux de données). Nous retrouvons, entre autres, les processeurs vectoriels et les processeurs graphiques (GPU, ou Graphics Processing Unit) dans cette architecture.

- L'architecture MISD (*Multiple Instruction on Single Data stream*, ou plusieurs instructions sur une seule donnée) : Dans cette architecture, plusieurs flux d'instructions sont appliqués à un même flux de données. Cette architecture n'est pas très populaire et est souvent utilisée dans le cadre de tests de tolérance aux pannes.
- L'architecture MIMD (*Multiple Instruction on Multiple Data stream*, ou plusieurs instructions sur plusieurs données) : Dans cette architecture, des flux d'instructions indépendants sont appliqués à des ensembles de données indépendants (plusieurs flux de données). La plupart des systèmes multiprocesseurs actuels sont basés sur cette architecture. Dans cette architecture, nous retrouvons entre autre la technique SPMD (Single Process Multiple Data stream, ou Single Program Multiple Data stream) qui est couramment employée pour réaliser du parallélisme (Darema *et al.*, 1988). Les tâches sont divisées et exécutées de façon concurrente sur différents processeurs. Cette technique est l'une des formes les plus courantes de la programmation parallèle.

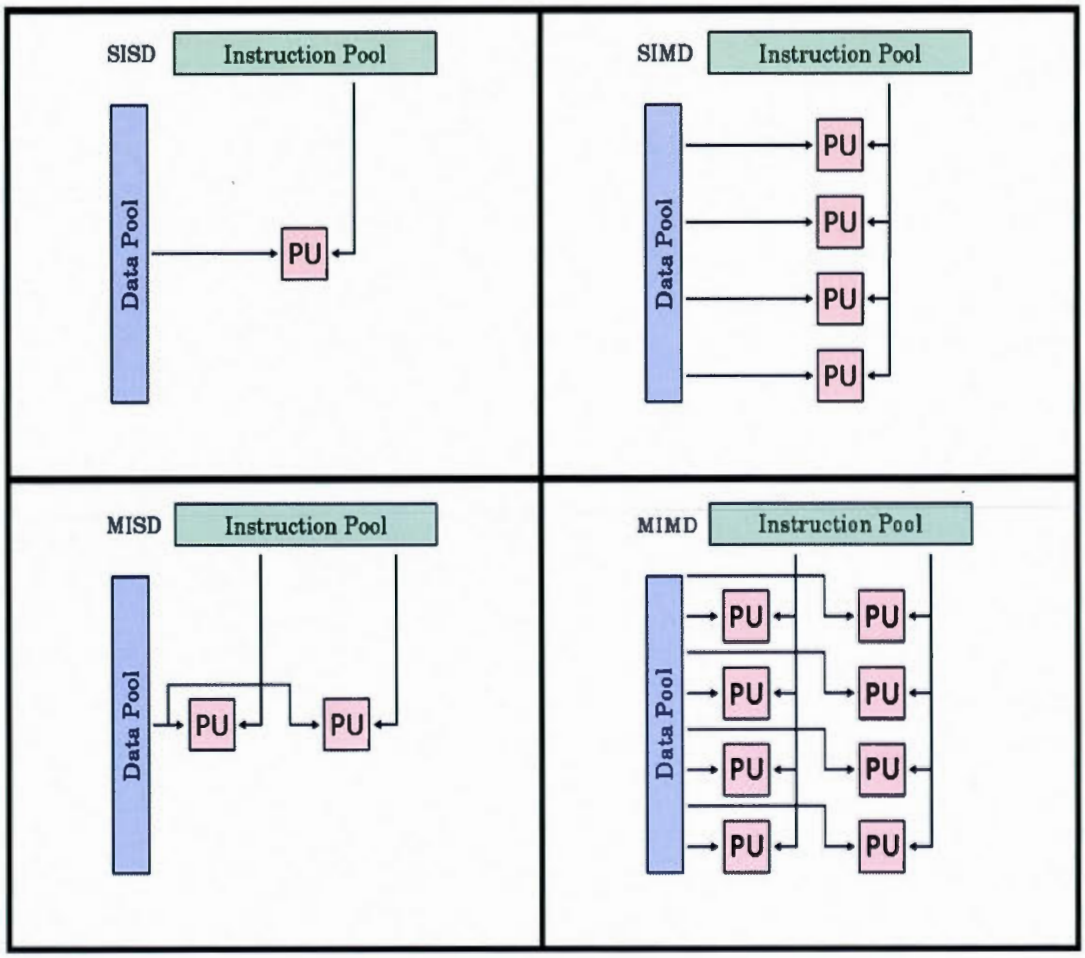


Figure 4.4 - Architecture de Flynn (1972).

Les systèmes informatiques multiprocesseurs peuvent aussi être divisés en deux groupes selon leurs liens avec la mémoire : les systèmes multiprocesseurs *centralisés* (ou *partagés*) et les systèmes multiprocesseurs *distribués*. Les systèmes centralisés possèdent une mémoire commune pour tous les processeurs qui sont interconnectés à travers un bus. Dans cette architecture, les processeurs communiquent les uns avec les autres en accédant à des variables partagées. Les systèmes multiprocesseurs distribués, comme son nom l'indique, sont des systèmes dans lesquels chaque processeur possède sa propre mémoire. Au niveau des systèmes distribués, les processeurs communiquent les uns avec les autres à travers l'échange de messages (Mattson *et al.*, 2005). Notons qu'il existe aussi des systèmes *hybrides* qui sont une combinaison de systèmes à mémoire distribuée et à mémoire partagée.

Pour assurer les communications, tout système multiprocesseur doit fournir aux processeurs un réseau d'interconnexion. Ce réseau est utilisé par les processeurs, soit pour communiquer les uns avec les autres, soit pour accéder aux données sauvegardées dans la mémoire. Il existe principalement deux types de réseaux d'interconnexions : le réseau à ligne unique (ou réseau partagé) et le réseau commuté. Le réseau à ligne unique permet l'échange d'un message à la fois. Avant l'envoi d'un message, le processeur doit s'assurer de la disponibilité du média. Le réseau commuté, quant à lui, permet de faire des échanges simultanés de messages. Il existe plusieurs topologies pour les réseaux commutés : le réseau en arbre binaire, le réseau en étoile, le réseau en hypercube, le réseau maillé, etc. (Mattson *et al.*, 2005).

Il existe deux approches principales de la programmation parallèle :

- Le parallélisme *implicite* : le système (le compilateur ou un autre programme) divise le problème à résoudre et assigne automatiquement les charges aux processeurs disponibles. Le parallélisme implicite est utilisé par des langages de programmation tels que HPF (High Performance Fortran) (Benkner et Zima, 1999).
- Le parallélisme *explicite* : le programmeur doit annoter manuellement le programme parallèle afin de repartir les différentes tâches aux processeurs disponibles. Le parallélisme explicite est utilisé par des modèles ou environnements de programmation parallèle tels que MPI (Gropp *et al.*, 1996) et OpenMP (Dagum et Menon, 1998; Chandra *et al.*, 2001).

Le développement de programmes parallèles comporte plusieurs difficultés liées principalement aux dépendances des données (Massingill *et al.*, 1999; Psarris et Kyriakopoulos, 2004; Johnson *et al.*, 2005; Dave *et al.*, 2009). Un programme séquentiel spécifie une certaine séquence d'actions à exécuter sur l'ordinateur et le compilateur parallèle essaye de regrouper ces actions de façon optimale. Tout regroupement et tout mélange de la séquence d'actions originale est permis tant que la fonction du programme reste intacte. Pour s'assurer que la fonction du programme demeure intacte, le compilateur doit trouver la structure des dépendances du programme. Cette structure de dépendances est déterminée

selon les différentes actions d'accès à la mémoire et aux structures de contrôle au niveau du code.

McGraw et Axelrod (1987) ont identifié quatre approches distinctes pour le développement d'applications parallèles :

- L'extension des compilateurs existants pour traduire des programmes séquentiels en programmes parallèles. Cette approche consiste à développer un compilateur parallèle qui peut détecter et exploiter le parallélisme de programmes existants, écrits dans des langages de programmation séquentiels. Par exemple, les compilateurs d'*Oracle Solaris Studio* (Oracle White Paper, 2010) utilisent cette approche pour faire du parallélisme.
- L'extension des langages de programmation séquentiels existants en rajoutant de nouvelles fonctionnalités qui permettent aux utilisateurs d'exprimer le parallélisme. Les langages de programmation séquentiels sont étendus en rajoutant des fonctionnalités qui permettent au programmeur la création et la terminaison de processus parallèles, ainsi que la synchronisation et la communication entre les processus. Des langages de programmation séquentiels tels que *ADA 2005* (Barnes, 2006) et *Java* (Arnold *et al.*, 2000) utilisent cette approche.
- L'ajout d'une nouvelle couche de parallélisation aux langages séquentiels existants. Cette nouvelle couche a pour but de contrôler la création et la synchronisation des processus, ainsi que la distribution des données entre les processeurs. Au niveau de cette approche, nous retrouvons, par exemple, la bibliothèque MPI (Gropp *et al.,* 1996) et l'API OpenMP (Dagum et Menon, 1998; Chandra *et al.*, 2001).
- La définition d'un nouveau langage de programmation parallèle et de son compilateur.

4.2 Environnements de programmation parallèle existants

Il existe plusieurs environnements, qui permettent de supporter l'exécution des applications de façon à exprimer le parallélisme. Les environnements de programmation parallèle fournissent les outils de base, les fonctionnalités du langage et les API nécessaires au développement d'un programme parallèle. Parmi ces environnements, nous pouvons citer : *Parallel Virtual Machine* (Geist *et al.*, 1994), qui existe depuis de nombreuses années, *Message Passing Interface* (Gropp *et al.*, 1996), *Open Multi-Processing* (Dagum et Menon, 1998; Chandra *et al.*, 2001), *MapReduce* (Ghemawat et Dean, 2004) introduit par *Google*, ainsi que les architectures logicielles et matérielles pour la mise en œuvre et la gestion des opérations exécutées sur le GPU (Fernando, 2004; Nvidia, 2010). Il faut aussi noter que certains langages de programmation séquentiels, tels que *ADA 2005* et *Java* entre autres, ont un ensemble de fonctions intégrées qui permet de faire de la programmation parallèle en spécifiant de manière explicite le traitement concurrent.

4.2.1 Environnement de programmation OpenMP

OpenMP (Open Multi-Processing) est une API définie et normalisée communément par un groupe de fournisseurs de matériels et de logiciels informatiques (Dagum et Menon, 1998; Chandra *et al.*, 2001). OpenMP offre un modèle portable et évolutif aux développeurs d'applications parallèles à mémoire partagée. C'est une série d'extensions de langages séquentiels implémentées sous la forme de directives au compilateur. L'API prend en charge les langages *C/C++* et *Fortran* sur des plates-formes multiples, dont *Unix* et *Windows* (Quinn, 2004).

Un programme OpenMP est exécuté par un processus unique. Ce processus unique active des processus légers (ou *threads*) à l'entrée d'une région parallèle et chaque processus léger exécute une tâche composée d'un ensemble d'instructions. Pendant l'exécution d'une tâche, une variable peut être lue et/ou modifiée en mémoire. Cette variable peut être définie dans la pile (espace mémoire local) d'un processus léger, on parle alors d'une variable privée. Elle peut aussi être définie dans un espace mémoire partagé par tous les processus légers, on parle alors d'une variable partagée (Chandra *et al.*, 2001).

Un programme OpenMP est une alternance de régions séquentielles et de régions parallèles (voir la Figure 4.5). Une région séquentielle est toujours exécutée par la tâche maîtresse, celle qui a le rang zéro. Une région parallèle, quant à elle, peut être exécutée par plusieurs tâches à la fois (voir la Figure 4.5). Les tâches partagent la charge de travail contenue dans la région parallèle. Le partage du travail consiste essentiellement à exécuter :

- une boucle par répartition des itérations entre les tâches;
- plusieurs sections de code, mais une seule par tâche;
- plusieurs occurrences d'une même procédure par différentes tâches.

Il est souvent nécessaire d'introduire des opérations de synchronisation entre les tâches concurrentes.

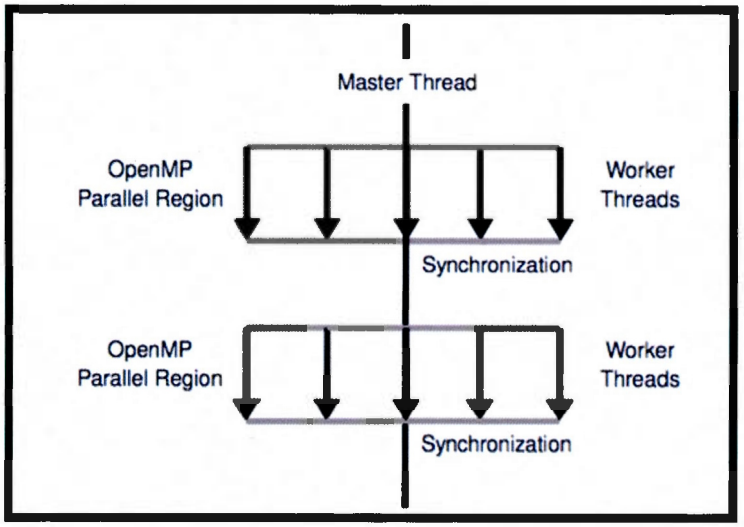


Figure 4.5 - Modèle d'exécution OpenMP (figure tirée d'Oracle White Paper, 2010).

4.2.2 Environnement de programmation GPU

Vers la fin des années 90, les cartes graphiques VGA (Video Graphics Array) sont devenues très sophistiquées et puissantes. Il y a eu un besoin de trouver un nouveau nom pour refléter cette nouvelle technologie en pleine évolution. Le terme GPU (Graphics Processing Unit) a été inventé en 1998. Ce terme a été fortement inspiré du CPU (Central Processing Unit).

Un GPU est un microprocesseur présent sur les cartes graphiques d'un ordinateur ou d'une console de jeux vidéo (Nvidia, 2010). Une partie du travail habituellement exécutée par le processeur principal peut ainsi être déléguée au processeur graphique. Au cours des dernières années, les processeurs graphiques ont évolué de façon fulgurante par rapport aux CPU (voir la Figure 4.6). Les GPU sont des architectures massivement parallèles qui sont beaucoup moins coûteux comparativement aux clusters et autres architectures parallèles.

Des comparaisons entre les GPU et les CPU sont souvent effectuées en termes de GFLOP/s (giga-flops par seconde) et GB/s (gigabytes par seconde). En termes de GFLOP/s, nous remarquons de façon générale que le GPU est beaucoup plus rapide que le CPU, que ça soit en simple ou en double précision. Au niveau de la bande passante, nous aboutissons aussi au même résultat allant en faveur du GPU (voir la Figure 4.6 a-b).

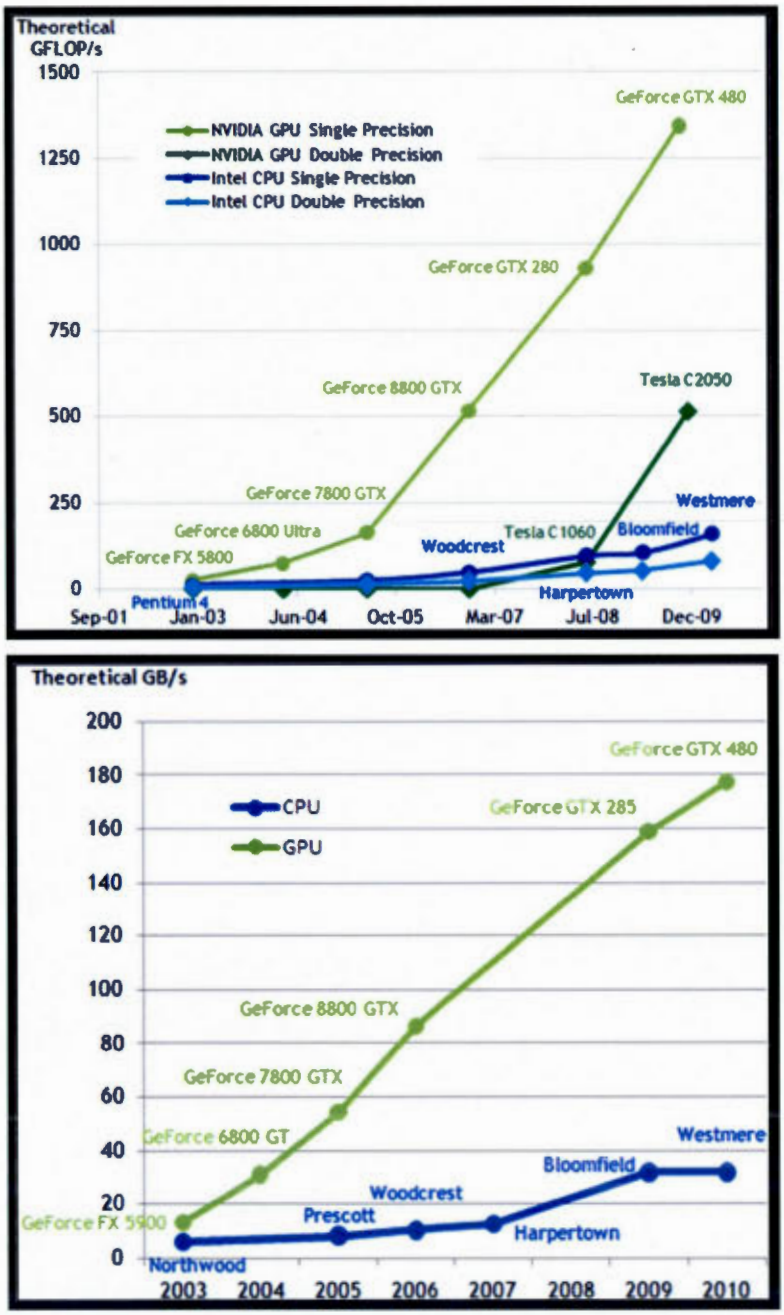


Figure 4.6 - Comparaison en termes d'opérations à virgule flottante par seconde (a), et en termes de bande passante de la mémoire (b) entre les CPU dual-core d'Intel et les GPU de Nvidia (figures tirées de Nvidia, 2010).

Les GPU actuels (Fernando, 2004) offrent d'énormes ressources aussi bien pour les traitements graphiques (but primaire du GPU) que pour les traitements non graphiques. Ils sont bien adaptés pour des problèmes qui peuvent être exprimés en traitements de données indépendantes en parallèle, car ils consacrent plus de transistors au traitement des données (voir la Figure 4.7). Cependant, les CPU, quant à eux, consacrent un nombre non négligeable de transistors pour conserver les données et contrôler les commandes à exécuter (voir la Figure 4.7).

Dans la Figure 4.7 :

- Les cases vertes représentent des *unités arithmétiques et logiques* (ALU, ou Arithmetic Logic Unit). Ces unités sont chargées d'effectuer les calculs.
- Les cases jaunes représentent des unités de *contrôle* ou séquenceurs qui commandent et contrôlent le fonctionnement du système.
- Les cases oranges représentent la *mémoire cache* (qui est une mémoire relativement petite et rapide) et la m*émoire dynamique à accès direct* ou DRAM (Dynamic Random Access Memory).

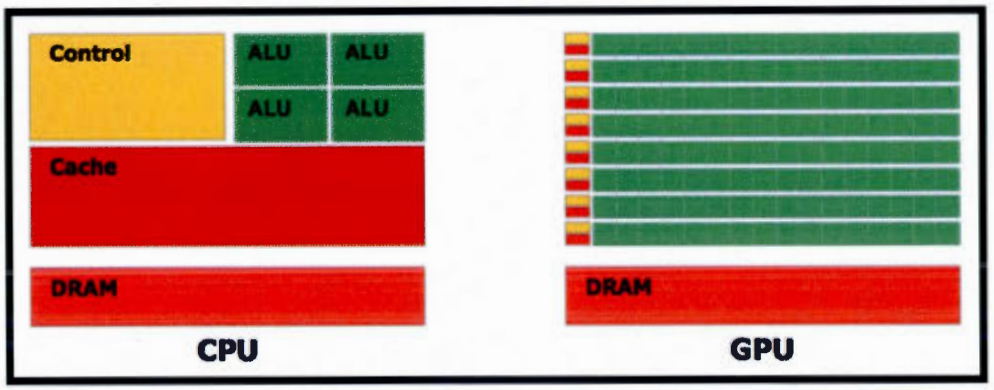


Figure 4.7 - Différences entre les architectures CPU et GPU (figure tirée de Nvidia, 2010).

Les GPU de Nvidia, l'un des plus grands fournisseurs de processeurs graphiques, peuvent être programmés à travers l'API CUDA (Compute Unified Device Architecture) qui est une architecture logicielle et matérielle pour la mise en œuvre et la gestion des opérations exécutées sur le GPU telles que le parallélisme de données. CUDA est destiné à supporter plusieurs langages de programmation tels que : *C/C++*, *Fortran*, etc.

Comme les GPU sont bien adaptés aux problèmes qui peuvent être exprimés en traitements de données indépendantes en parallèle, nous avons exploré cette piste afin de développer des méthodes de transformation d'algorithmes séquentiels en algorithmes parallèles exécutables sur les GPU. Au début de notre projet doctoral, nous avons effectué des tests impliquant les GPU de *Nvidia GeForce 9800 GX2* sur des programmes parallèles bioinformatiques. Ces tests n'ont pas donné de résultats concluants du fait que la mémoire locale du GPU n'était pas assez volumineuse pour le traitement efficace de grands problèmes bioinformatiques. Cependant, il faut noter que l'API CUDA était encore à sa première version et contenait de nombreux bogues. Mentionnons que la méthode de parallélisation à travers les GPU est basée sur l'architecture SIMD, alors que la méthode de parallélisation à travers MPI, que nous avons adoptée dans le cadre de ce projet doctoral (voir la section 4.2.4), est basée sur l'architecture MIMD.

4.2.3 Environnement de programmation PVM

Le modèle de programmation PVM (Parallel Virtual Machine) est basé sur le concept de machine virtuelle. Une machine virtuelle, comme son nom l'indique, est composée d'une ou de plusieurs machines hôtes qui communiquent les unes avec les autres à travers le protocole standard TCP/IP (Transmission Control Protocol / Internet Protocol). Les machines hôtes peuvent être une collection de postes de travail hétérogènes, des superordinateurs, des PC (Personal Computers), etc. Du point de vue du programmeur, la machine virtuelle est un ordinateur unique disposant d'une grande mémoire. Dans le modèle PVM, la machine virtuelle peut être constituée d'un nombre illimité d'hôtes et ces derniers peuvent être ajoutés ou supprimés dynamiquement. Les processus qui s'exécutent sur les machines parallèles sont appelés les tâches. Le modèle PVM assure l'exécution de plusieurs tâches au niveau d'un seul hôte et leur communication les unes avec les autres. Ces tâches peuvent aussi être gérées de manière dynamique (Geist *et al.*, 1994).

Les techniques de calculs parallèles utilisées par PVM, sont maintenant employées dans pratiquement toutes les disciplines scientifiques. La mise à disposition des codes sources de PVM dans le domaine public, le développement des réseaux de télécommunications et la

présence sur toutes sortes de plates-formes ont été des éléments déterminants pour le succès de ce modèle.

4.2.4 Environnement de programmation MPI

4.2.4.1 Introduction

MPI (Message Passing Interface) est une bibliothèque standard de passage de messages (Gropp *et al.*, 1996). L'échange (ou passage) de messages est une méthode par laquelle les données de la mémoire d'un processeur sont copiées dans la mémoire d'un autre processeur. Le modèle de passage de messages est habituellement utilisé pour assurer la communication au niveau des machines parallèles à mémoire distribuée. Cependant, il existe des versions pour les machines parallèles à mémoire partagée et les machines parallèles hybrides. La bibliothèque standard de passage de messages est constituée d'une collection de routines servant à faciliter la communication entre les processeurs d'une machine parallèle. La Figure 4.8 montre une modèle d'exécution typique de l'environnement MPI.

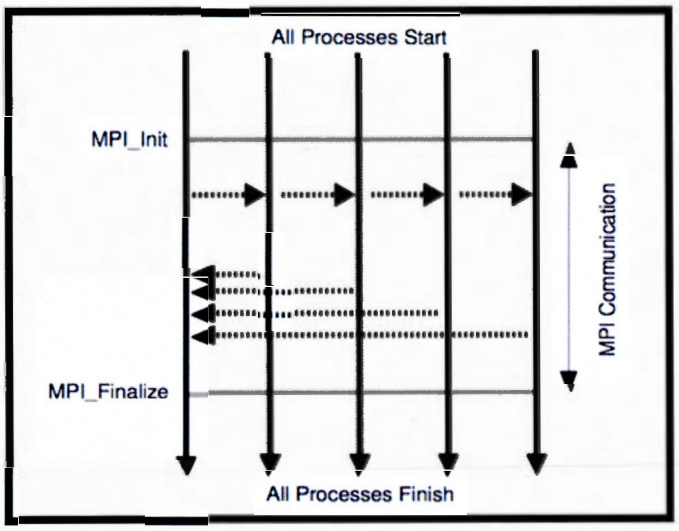


Figure 4.8 - Modèle d'exécution MPI (figure tirée d'Oracle White Paper, 2010).

Les routines de communication peuvent être de type point à point ou collectives. Les routines point à point permettent à deux processus d'échanger une donnée, à travers des fonctions telles que *MPI_Send* et *MPI_Recv* (voir la section 4.2.4.2). Les routines collectives, quant à elles, impliquent un ensemble de processeurs. Par exemple, il est possible d'envoyer une même donnée à tous les processeurs à travers la fonction *MPI_Bcast* (voir la section 4.2.4.2).

Le but principal du projet MPI était de concevoir une bibliothèque complète qui assure :

- la communication de haut niveau et de bas niveau;
- la portabilité : elle est très importante avec l'évolution et les changements rapides des machines parallèles et de la technologie;
- la modularité (au niveau du développement des différentes bibliothèques);
- la gestion des processus;
- la gestion des groupes de processus.

Il existe des implémentations de MPI pour les systèmes d'exploitation *Unix / Linux* et *Windows*. Ces implémentations supportent les programmes écrits en *C/C++*, *Java*, *Fortran*, *Python* et *OCaml*.

MPI résulte des travaux de nombreuses équipes de recherche qui ont été réalisés entre les années 1992 et 1994. Une première version standard *MPI-1* a été dévoilée et rendue disponible en mai 1994. Ensuite, la version *MPI-2* (Gropp *et al.*, 1999) a été développée. Cette version traite des sujets qui vont au delà de ceux de la première spécification de MPI. Aujourd'hui, les différentes implémentations de MPI sont une combinaison des fonctionnalités incluses dans *MPI-1* et dans *MPI-2*. Toutefois, peu d'implémentations incluent la totalité des fonctionnalités de *MPI-1* et de *MPI-2*. Notons aussi, dans un dernier temps, que MPI utilise la technique SPMD (Darema *et al.*, 1988).

4.2.4.2 Échanges de messages et routines MPI

La bibliothèque d'échange de messages, MPI, utilise des objets appelés *communicateurs* ou *groupes* pour définir l'ensemble des processeurs qui peuvent communiquer les uns avec les autres (voir la Figure 4.9). La plupart des routines MPI exigent que la spécification du *communicateur* soit passée en argument. Le *communicateur* par défaut est la constante *MPI_COMM_WORLD* qui inclut tous les processeurs MPI. La création de nouveaux communicateurs est permise par la bibliothèque d'échange de messages (voir la Figure 4.9).

Dans un *communicateur* (voir la Figure 4.9), chaque processeur est identifié de façon unique par un entier qui lui est assigné par le système lors de l'initialisation de l'environnement. Les identifiants des processeurs, aussi appelés rangs des processeurs, sont contigus et commencent à zéro. Les rangs sont généralement utilisés pour spécifier la source et la destination des messages. Ils peuvent aussi être utilisés pour le contrôle de l'exécution de certaines parties du programme par certains processeurs seulement (par exemple, si *rang* = 0 alors exécuter la suite d'instructions suivante).

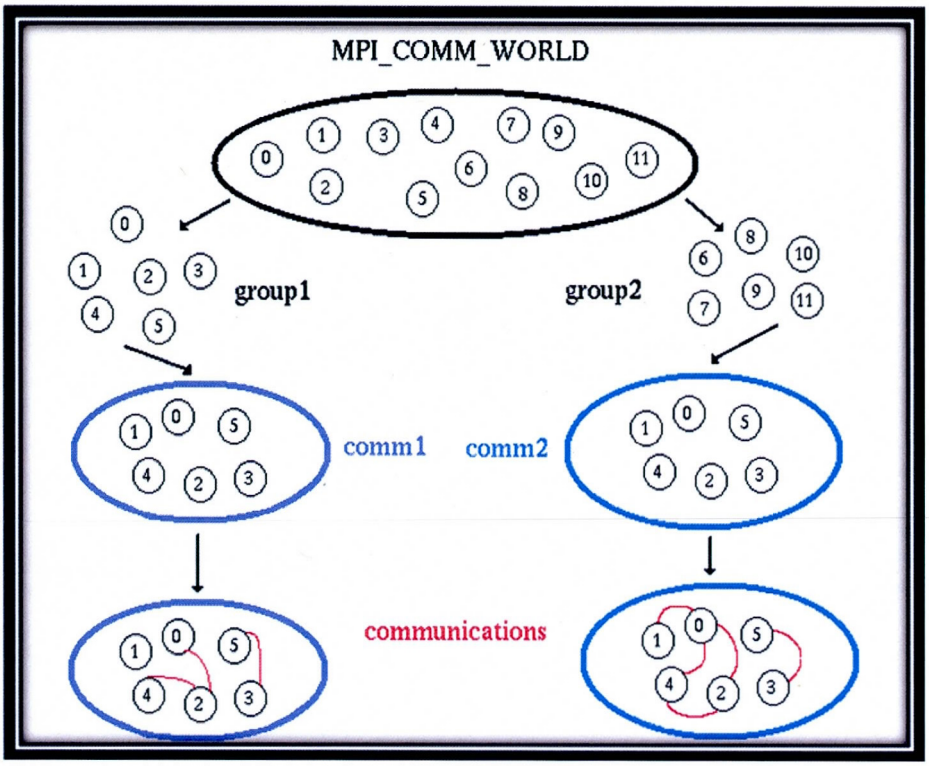


Figure 4.9 - Un exemple d'un communicateur (ou groupe) MPI (figure tirée du site : http://einspem.upm.edu.my/INSPEM/mpi.jsp).

Les routines MPI peuvent être divisées en trois grandes catégories selon leurs fonctions :

- Les routines de gestion de l'environnement d'exécution sont utilisées afin d'atteindre des buts spécifiques tels que l'initialisation et la terminaison de l'environnement MPI, le questionnement de l'environnement, tel que son identité, etc. Les routines de gestion d'environnement les plus couramment utilisées sont les suivantes : *MPI_Comm_size*, *MPI_Comm_rank*, *MPI_Abort*, *MPI_Initialized*, *MPI_Finalize*, *MPI_Get_processor_name*, *MPI_Wtime* et *MPI_init*.

- Les routines de communications *point à point* impliquent l'échange de messages entres deux processeurs différents. Le premier processeur exécute l'opération d'envoi du message, tandis que le second exécute l'opération de réception du message correspondant (voir la Figure 4.10).

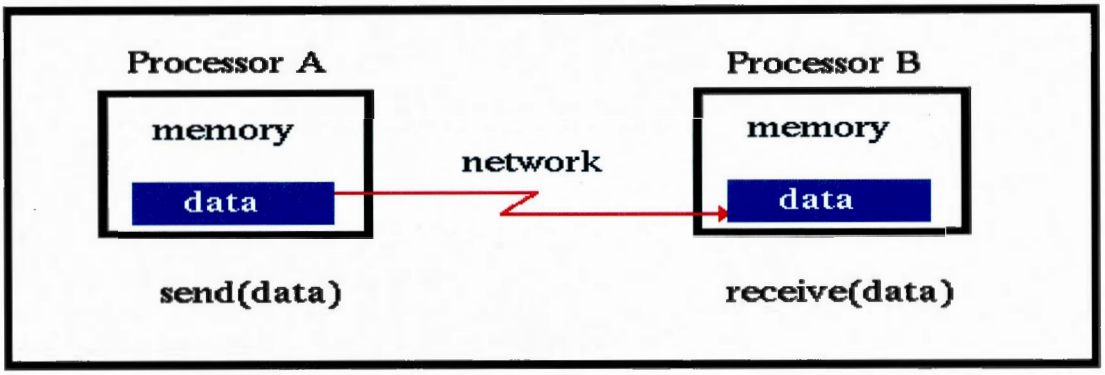


Figure 4.10 - Un exemple d'envoi et de réception de messages à travers MPI (figure tirée du site : http://einspem.upm.edu.my/INSPEM/mpi.jsp).

La plupart des routines de communication *point à point* peuvent être utilisées en mode *bloqué* ou *débloqué*. Les routines les plus communes sont les suivantes : *MPI_Send*, *MPI_Recv*, *MPI_Isend* et *MPI_Irecv*.

- Les routines de communications collectives impliquent tous les processeurs d'un communicateur donné (par exemple, tous les processeurs qui appartiennent au communicateur par défaut *MPI_COMM_WORLD*). Les communications collectives peuvent être de trois types : la synchronisation (les processeurs attendent que tous les membres du *groupe* aient atteint le point de synchronisation), les *déplacements* de données (ce sont les opérations des types *broadcast*, *distribution* et *regroupement* des données, etc.) et les opérations collectives (un membre du groupe collecte les données provenant de tous les autres processeurs du *groupe* et effectue une opération sur les données collectées). Toutes les opérations collectives sont bloquantes. Les routines de communication collectives les plus communes sont les suivantes : *MPI_Barrier*, *MPI_Bcast*, *MPI_Scatter*, *MPI_Gather* et *MPI_Reduce*.

Pour des raisons de portabilité, MPI a défini ses types de données élémentaires et permet de définir des types dérivés en utilisant *MPI_PACKED*. La correspondance entre les types de données élémentaires de MPI et du langage *C* est présentée dans l'Annexe D (voir le Tableau D.1).

À part les routines citées précédemment, MPI inclut des routines qui permettent de définir ses propres structures de données en se basant sur ses types élémentaires. Ces structures sont appelées types de données dérivés. La totalité des routines de MPI pour l'environnement de programmation *C* est présentée dans l'Annexe D (voir le Tableau D.2).

4.2.4.3 Les raisons du choix de MPI dans le cadre de notre projet doctoral

Nous avons utilisé la bibliothèque MPI dans le cadre de notre projet pour les raisons suivantes :

- La standardisation : MPI est la seule bibliothèque d'échange de messages qui peut être considérée comme standard. Cette bibliothèque remplace progressivement toutes les anciennes bibliothèques d'échange de messages;
- La portabilité : Aucune nécessité de faire une modification sur le code à la suite d'un changement de plateforme qui supporte le standard MPI;
- La disponibilité : Plusieurs mises en œuvre de MPI sont disponibles (dans les domaines industriel et public). Nous avons utilisé la version MPICH2 (release MPICH2-1.4.1p1) de MPI qui est disponible à l'adresse URL suivante : http://www.mcs.anl.gov/research/projects/mpich2;
- L'existence de plusieurs programmes bioinformatiques parallélisés à travers la bibliothèque MPI (voir le Tableau 1.2 pour plus d'informations).
- La disponibilité au laboratoire de recherche en bioinformatique à l'UQAM d'un cluster (grappe de processeurs) avec un système d'exploitation *Linux* : *CentOS release 5.7* (*Final*). Ce cluster dispose de 16 lames possédant deux processeurs de 3.2 GHZ chacun. Il utilise une architecture distribuée. Mentionnons que MPI est principalement employé dans les modèles de programmation parallèle à mémoire distribuée. Ce cluster a été acquis par mon directeur de recherche, Monsieur Vladimir Makarenkov, dans le cadre d'un projet supporté par le FCI (Fondation Canadienne pour l'Innovation).

4.2.5 Environnements de programmation hybrides

Il existe aussi des environnements de programmation hybrides. Les environnements hybrides sont, de façon générale, une association des modèles de parallélisation à mémoire distribuée et partagée. Dans ce cas, nous pouvons utiliser la bibliothèque d'échange de messages, MPI, pour effectuer du parallélisme entre les différentes nœuds du système, alors que l'API OpenMP peut être est utilisée pour le parallélisme à l'intérieur d'un nœud. Notons que la mémoire partagée d'un système est mieux exploitée par OpenMP et ne nécessite pas une duplication (contrairement à MPI).

Nous pouvons aussi citer les environnements hybrides tels que l'environnement CPUGPU (Lee *et al.*, 2009). Ce type d'environnement est de plus en plus en vogue en bioinformatique (*e.g.*, logiciel ProtTest introduit par Darriba *et al.*, 2011). Cependant, dans ce genre de systèmes, les programmes développés sont moins portables à cause de la spécificité des architectures.

4.3 Métriques de performance

Dans le cadre de notre projet doctoral, nous examinons également diverses métriques qui permettent d'estimer les performances d'un algorithme. Les notions qui sont abordées sont les suivantes : le temps d'exécution, le coût, l'accélération et l'efficacité. Les sous-sections suivantes décrivent ces métriques.

4.3.1 Temps d'exécution

Le temps d'exécution peut être défini comme l'intervalle de temps écoulé entre le début et la fin de l'exécution d'un programme. Le temps séquentiel est l'intervalle de temps écoulé entre le début et la fin de l'exécution du programme séquentiel. Le temps parallèle, quant à lui, est l'intervalle de temps écoulé entre le début et la fin d'un programme parallèle exécuté sur un nombre de processeurs déterminé.

4.3.2 Coût

Le coût d'un programme parallèle tient compte à la fois du temps d'exécution, ainsi que du nombre de processeurs utilisés. L'ajout de processeurs ou de toutes nouvelles composantes est toujours couteux (*i.e.* : coût d'achat, coût d'installation, coût de fonctionnement, etc.).

Soient un programme A utilisé pour résoudre de façon parallèle un problème de taille n en un temps $T_P(n)$ et $p(n)$ le nombre maximum de processeurs requis par A. Le coût de A est alors défini comme suit :

$$C(n) = p(n) * T_P(n).$$

4.3.3 Accélération

L'accélération permet de déterminer de combien le programme parallèle est plus rapide que le programme séquentiel équivalent. On distingue deux formes d'accélération : l'accélération *relative* et l'accélération *absolue*.

L'accélération *relative* peut être définie comme le rapport entre le temps d'exécution du programme parallèle sur un seul processeur $T_P(1)$ et le temps d'exécution du même programme parallèle sur n processeurs $T_P(n)$:

$$\textit{Accélération relative} = T_P(1) / T_P(n).$$

L'accélération *absolue*, quant à elle, peut être définie comme le rapport entre le temps d'exécution du meilleur programme séquentiel possible T_S^* et le temps d'exécution parallèle sur n processeurs $T_P(n)$:

$$\textit{Accélération absolue} = T_S^* / T_P(n).$$

4.3.4 Efficacité

L'efficacité nous indique dans quelle mesure les divers processeurs sont réellement utilisés ou pas. L'efficacité d'un programme est le rapport entre le temps d'exécution du meilleur programme séquentielle possible T_S^*et le coût d'exécution du programme parallèle avec n processeurs $T_P(n)$:

$$Efficacité = T_S^* / (T_P(n) * n).$$

En d'autres termes, l'efficacité est obtenue en divisant l'accélération absolue par le nombre de processeurs n :

$$Efficacité = Accélération / n.$$

4.3.5 Loi d'Amdahl

La loi d'Amdahl (Amdahl, 1967), énoncée par Gene Amdahl, exprime le gain de performance qu'on peut attendre d'un ordinateur en améliorant une composante de sa performance. Sous sa forme générale elle indique que le gain de performance est égal au temps d'exécution d'une tâche complète sans l'amélioration divisé par le temps d'exécution de la même tâche avec l'amélioration, en négligeant le surcroit d'activité liée à la gestion du parallélisme lui-même. La formule d'Amdahl est la suivante :

$$R = \frac{1}{(1-s)+\frac{s}{N}},$$

où R représente le gain de performance maximal que le programme parallèle peut apporter, N représente le nombre de processeurs et s la portion parallélisable du programme.

La loi d'Amdahl ne tient pas compte du fait que la mise en place du parallélisme et les échanges de messages peuvent aussi influencer la durée d'exécution d'un programme.

4.4 Modèles de conception de programmes parallèles

4.4.1 Introduction

Pour la parallélisation de programmes séquentiels, nous nous sommes inspirés des concepts présentés dans les quatre espaces de conception de programmes parallèles, introduits par Massingill, Mattson et Sanders (MMS). Ces quatre espaces sont les suivants : la recherche de la concurrence (*Finding Concurrency*), la structuration de l'algorithme (*Algorithm Structure*), les structures de soutien (*Supporting Structures*) et l'implémentation des mécanismes (*Implementation Mechanisms*). Ces quatre espaces (voir la Figure 4.11) forment une hiérarchie linéaire, avec la recherche de la concurrence au sommet et l'implémentation des mécanismes tout en bas (Massingill *et al.*, 2000; Mattson *et al.*, 2005). Ces quatre espaces forment aussi ce que l'on appelle un langage de modèles ou un langage de patrons « *pattern language* ».

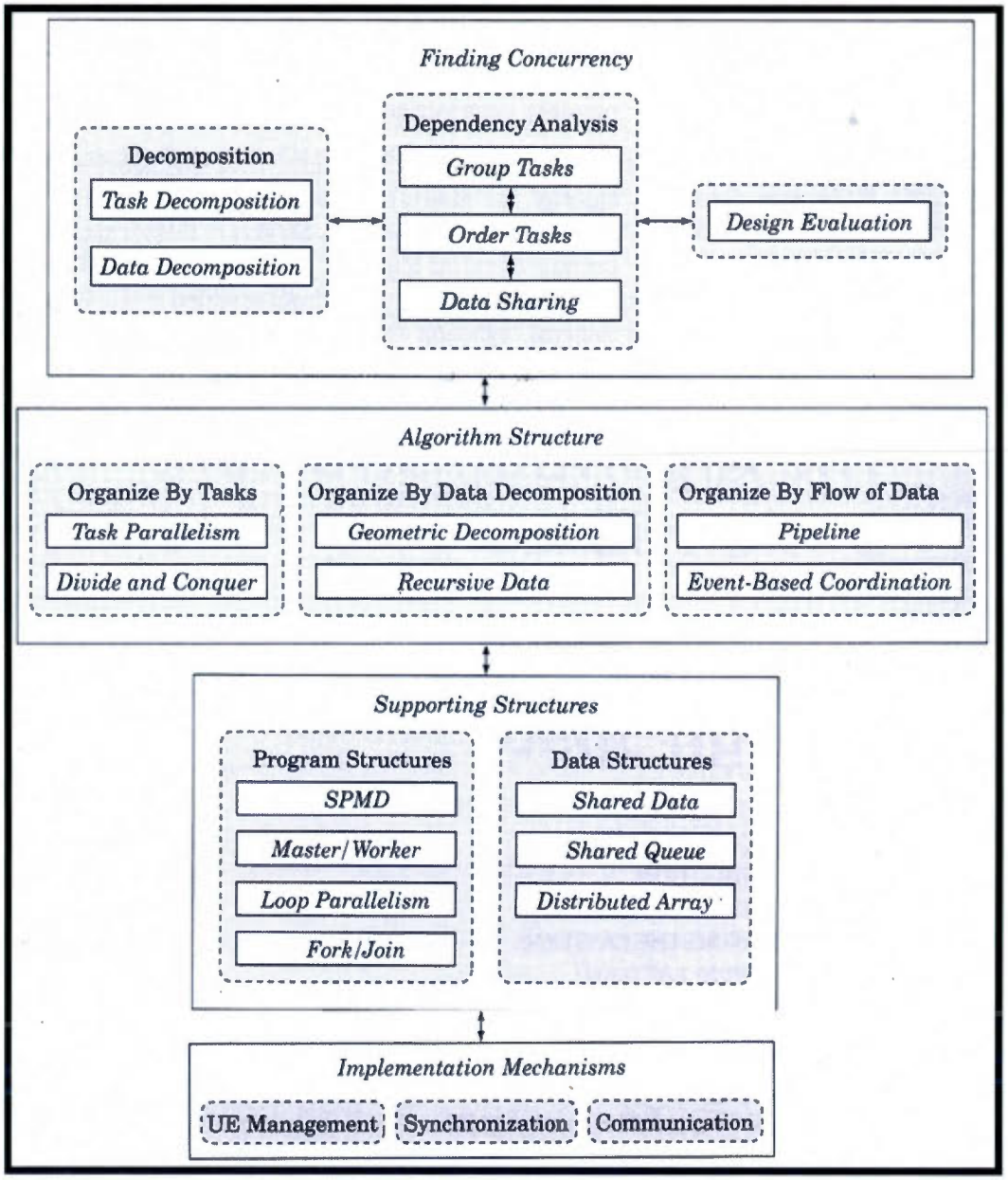


Figure 4.11 - Espaces de conception de programmes parallèles de MMS (figure tirée de Mattson *et al.*, 2005).

Un langage de patrons peut être défini comme une collection de patrons de conception « *design pattern* » fortement structurée afin d'offrir une méthodologie de conception. Un patron de conception correspond à la définition d'une solution de qualité dédiée à la résolution d'un problème fréquent dans un domaine spécifique. Le but général de l'introduction des patrons de conception est de minimiser les interactions entre les différentes classes (ou modules) d'un même programme. L'avantage de ces patrons est de diminuer le temps nécessaire au développement d'un logiciel et d'augmenter la qualité du résultat, notamment en appliquant des solutions existantes à des problèmes courants de conception. Un langage de patrons permet aussi, de façon générale, de guider le concepteur ou le développeur à travers toutes les étapes de la conception. À chaque étape, la solution appropriée est sélectionnée puis l'étape suivante est amorcée (Massingill *et al.*, 2000).

La recherche de concurrence est le premier espace de conception de la hiérarchie de MMS (voir la Figure 4.11). Cet espace a pour objectif d'exposer le problème principal en sous-problèmes pouvant être traités de manière simultanée ou concurrente. Cet espace peut être divisé en trois parties (Massingill *et al.*, 2000), qui sont la décomposition du problème, l'analyse des dépendances et l'évaluation de la conception :

- La décomposition du problème :

  La première étape dans la conception d'un algorithme parallèle est la décomposition du problème en éléments pouvant être exécutés de manière simultanée. Il existe deux stratégies de décomposition : la décomposition en fonction des *données* et la décomposition en fonction des *tâches* (Massingill *et al.*, 1999). La décomposition en fonction des *tâches* peut être vue comme un ensemble d'instructions, qui est divisé en séquences d'instructions pouvant être exécutées au même moment. Ces séquences d'instructions sont appelées *tâches*. Afin d'obtenir des résultats acceptables, c'est-à-dire des résultats qui justifient l'emploi du parallélisme, ces *tâches* doivent être fortement indépendantes. Le principal défi est de trouver une bonne, voire la meilleure, décomposition des *tâches*. La décomposition en fonction des *données,* quant à elle, se focalise sur la distribution en différents fragments, des *données* requises par les *tâches*. Les traitements associés aux différents fragments de données

ne seront efficaces que si ces fragments peuvent être traités de façon indépendante les uns des autres.

- L'analyse des dépendances (*Dependency Analysis*) :

Une fois les différentes entités de la décomposition du problème identifiées, le modèle d'analyse des dépendances (Massingill *et al.*, 1999) permet de comprendre la façon dont toutes ces entités dépendent les unes des autres (les dépendances des données, les contraintes d'ordonnancement des tâches, etc.).

- L'évaluation de la conception (*Design Evaluation*) :

L'évaluation de la conception est une étape de consolidation qui a pour but d'évaluer les résultats obtenus suite à l'application des patrons de l'espace de recherche de concurrence (Massingill *et al.*, 1999; Massingill *et al.*, 2000) (voir la Figure 4.11). Cette étape peut être vue comme une vérification avant le passage à l'espace suivant, qui est la structuration de l'algorithme.

Le second espace de conception de MMS (voir la Figure 4.11) a pour but de structurer l'algorithme de façon à prendre pleinement avantage du traitement concurrent. C'est une étape de raffinement de la solution obtenue lors de l'étape précédente (la recherche de la concurrence). La conception de la structure d'algorithme doit suivre l'un des modèles suivants : le modèle de parallélisme de tâches, le modèle diviser pour régner, le modèle de décomposition géométrique, le modèle de données récursives, le modèle de pipeline ou le modèle de coordination basé sur les évènements (Hendren et Nicolau, 1990, Mattson *et al.*, 2005). Les deux premiers modèles cités permettent d'organiser l'algorithme en fonction des *tâches*, les deux modèles suivants se retrouvent dans la catégorie de l'organisation de l'algorithme en fonction de la décomposition de *données* et les deux derniers modèles sont dans la section de l'organisation de l'algorithme en fonction du *flux de données*. Le but principal de cette étape est d'identifier les modèles qui sont les plus appropriés à la résolution du problème posé.

Le troisième espace de conception de MMS (voir la Figure 4.11) définit les organisations architecturales et les structures logicielles qui supporteront le programme parallèle. Ces structures peuvent être divisées en deux groupes : les structures axées sur les

tâches et les structures axées sur les données (Massingill *et al.*, 2000; Mattson *et al.*, 2005). Au niveau des structures axées sur les *tâches* du programme, nous retrouvons l'architecture *SPMD* (*Single Program, Multiple Data*), l'architecture *maitre / travailleur*, l'architecture de *parallélisme de boucle* et l'architecture *« fork / join »*. Au niveau des structures axées sur les *données*, nous retrouvons l'architecture de *données partagées*, l'architecture de *queues partagées* (*Shared Queue*) et l'architecture des *tableaux distribués* (*Distributed Array*).

Le dernier espace de conception de MMS (voir la Figure 4.11) peut être subdivisé en trois catégories : la gestion des unités d'exécution, la synchronisation et les communications (Massingill *et al.*, 2000; Mattson *et al.*, 2005). La gestion des unités d'exécution regroupe les opérations de création, de terminaison et de gestion des processus exécutés lors du traitement parallèle. La synchronisation consiste à appliquer les contraintes sur l'ordonnancement des tâches traitées sur des unités d'exécution différentes de manière à s'assurer de la validité du résultat final. La synchronisation s'effectue généralement en utilisant des concepts de barrières. Enfin, les communications incluent tous les échanges d'informations entre les différentes unités d'exécution.

4.4.2 Application des espaces de conception de MMS à un algorithme bioinformatique pratique

Rappelons, tout d'abord, que la plupart des algorithmes bioinformatiques sont caractérisés par des traitements de données indépendantes et volumineuses (voir le chapitre I, section 1.6). Nous débutons donc par l'identification des caractéristiques importantes et principales des algorithmes bioinformatiques séquentiels que nous considérons dans cette thèse de doctorat. Par exemple, dans le cadre de la résolution des problèmes portant sur la recherche d'un couple d'espèces ayant un regroupement optimal (ce genre de problème caractérise de nombreux algorithmes bioinformatiques), nous pouvons identifier la tâche principale comme étant le calcul des valeurs d'un ou de plusieurs critères d'optimisation. En généralisant cet exemple, nous pouvons dire que les algorithmes bioinformatiques contiennent dans la plupart du temps une tâche *principale*, ou *maîtresse*, qui traite de larges ensembles de données biologiques indépendantes. Une décomposition du problème à résoudre en fonction des *données* serait donc préférable dans ce cas, plutôt qu'une décomposition en fonction des

tâches (voir la Figure 4.11). Une autre caractéristique importante d'une tâche principale d'un algorithme bioinformatique est qu'elle s'effectue généralement dans des boucles *for* imbriquées. Nous nous sommes alors focalisés sur la parallélisation des algorithmes bioinformatiques contenant ce genre de boucles.

Considérons un exemple concret : le programme implémentant l'algorithme NJ (Saitou et Nei, 1987, voir la section 1.3.1 du chapitre I pour plus de détails sur cet algorithme). Nous pouvons identifier la tâche principale comme étant les étapes 1 et 2 de l'algorithme. Ces deux étapes consistent à calculer la valeur du critère d'optimisation choisi pour tous les couples d'espèces et à sélectionner le couple optimal en fonction des valeurs de ce critère. Cette tâche est implémentée généralement à travers des boucles *for* imbriquées. L'exécution d'une telle tâche pourrait se faire en parallèle. Même si la détection des tâches principales est souvent facile, voir évidente, le développement d'une version parallèle, qui prend pleinement avantage du parallélisme, constitue un processus délicat. Afin de faciliter le processus de parallélisation, nous pouvons utiliser les patrons introduits dans les espaces de conception de MMS. Le but principal étant de guider le développeur du logiciel pour minimiser le temps nécessaire au développement du programme parallèle. Pour ce faire, nous devons choisir à chaque étape la solution la plus appropriée avant de passer à l'étape suivante (Massingill *et al.*, 2000). Les principaux patrons des espaces de conceptions de MMS utilisés dans notre exemple sont encerclés dans la Figure 4.12.

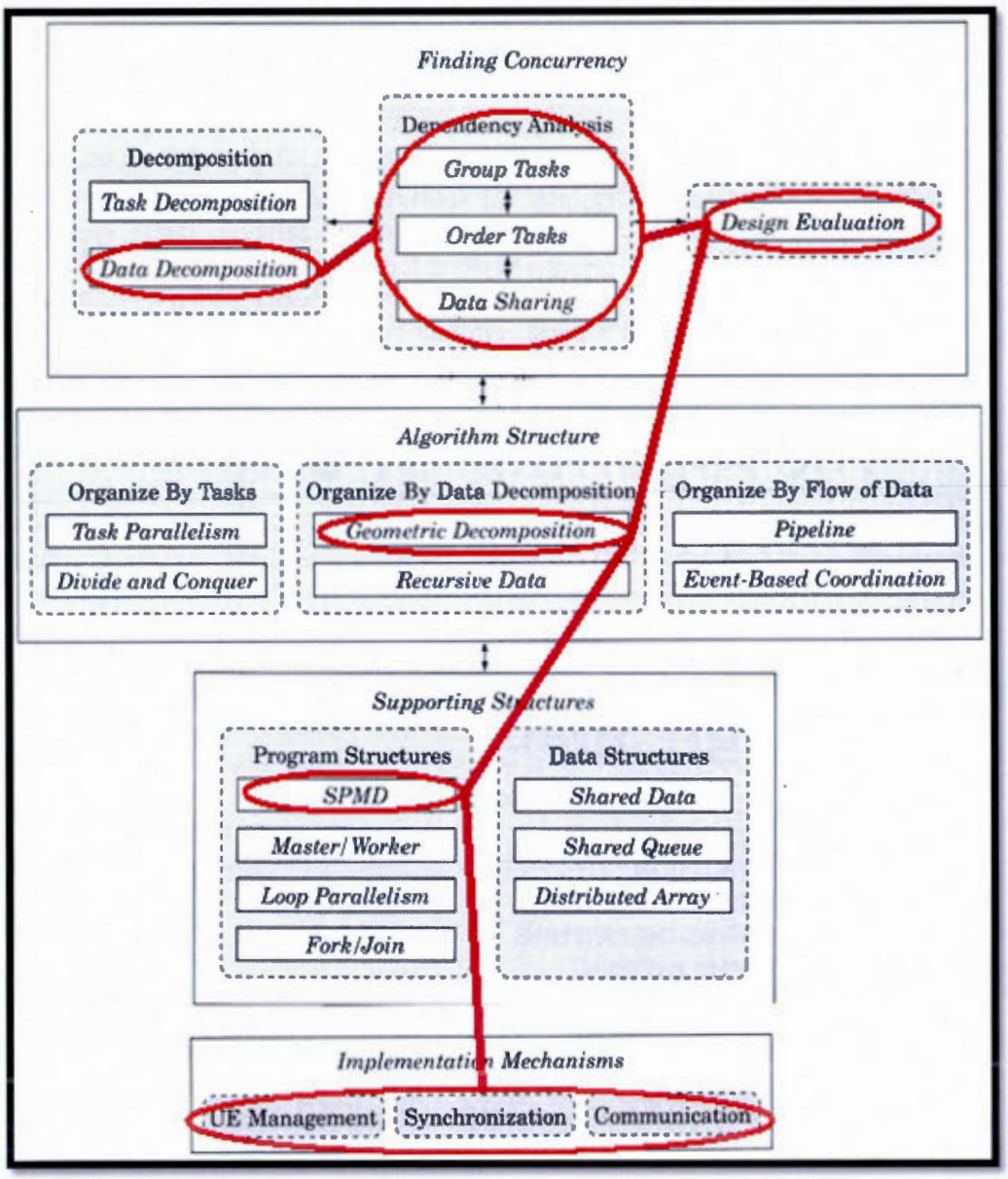


Figure 4.12 - Les patrons retenus (encerclés) pour la parallélisation de l'algorithme NJ (Saitou et Nei, 1987).

4.4.2.1 Espace de la recherche de concurrence

Au niveau de la décomposition du problème, les traitements des données associés aux différentes espèces analysées par l'algorithme NJ (Saitou et Nei, 1987) sont indépendants les

uns des autres. Les structures associées à ces données sont *linéaires* (c'est-à-dire des structures de données telles que les tableaux, les matrices, etc.). Quand nous avons des calculs ou des traitements indépendants de données, la décomposition du programme en fonction des données est la plus appropriée. C'est l'une des principales caractéristiques des structures de données associées (structures *linéaires*) aux algorithmes séquentiels bioinformatiques.

L'analyse des dépendances des données peut être effectuée soit par l'utilisateur qui a de bonnes connaissances de la structure du programme à paralléliser, soit à travers des méthodes d'analyses des dépendances (Wolfe et Banerjee, 1987; Banerjee, 1988; Psarris et Kyriakopoulos, 1999; Psarris et Kyriakopoulos, 2004).

Avant de passer à l'espace suivant, l'utilisateur devra effectuer l'évaluation de la conception permettant d'estimer et de prévoir les résultats qui seront obtenus après la parallélisation.

4.4.2.2 Espace de la structure de l'algorithme

Sachant que la décomposition en fonction des *données* a été recommandée lors la décomposition de l'algorithme NJ, nous pouvons appliquer soit un modèle de décomposition géométrique, quand le problème est traité de façon linéaire, soit un modèle de décomposition récursive dans les autres cas (voir la Figure 4.13). En utilisant la décomposition géométrique, nous pouvons réduire le problème à résoudre à des composantes concurrentes en découpant la structure de données en sous-structures d'une manière analogue à la division d'une région géométrique. Dans la Figure 4.13, nous avons encerclé la suite des étapes de MMS qui mènent à une décomposition géométrique des données.

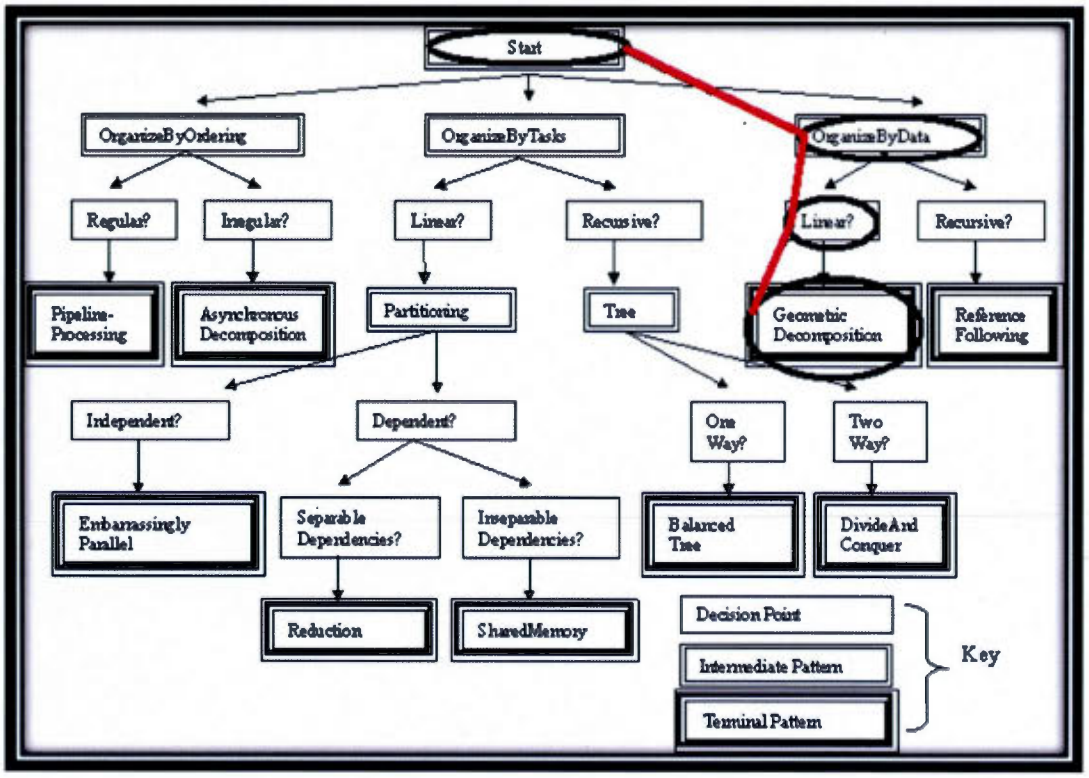


Figure 4.13: Arbre de décision pour la sélection des patrons du deuxième espace de conception de MMS (figure tirée de Mattson *et al.*, 2005). La suite des étapes de MMS que nous avons encerclées mène à une décomposition géométrique des données dans le cas de l'algorithme NJ (Saitou et Nei, 1987).

4.4.2.3 Espace de la structure de soutien

Comme nous avons utilisé la bibliothèque d'échange de messages MPI, la structure de soutien qui lui sera associée est l'architecture SPMD (Darema 1988; Mattson *et al.*, 2005). Dans cette architecture, chaque instance du programme exécute une partie du traitement à effectuer et les résultats sont combinés par la suite.

4.4.2.4 Espace de l'implémentation de mécanismes

Au niveau de l'implémentation de mécanismes, la gestion des unités d'exécution, la synchronisation et les communications sont effectuées à travers les routines qui sont incluses

dans la bibliothèque d'échange de messages MPI (voir l'Annexe D.1 pour plus de détails sur les routines MPI).

4.5 Revue de la littérature sur la parallélisation automatique

Ils existent plusieurs programmes bioinformatiques dont les versions parallèles ont été développées (voir le Tableau 1.2). Cependant, le développement de ces versions parallèles n'a été fait automatiquement. Les premiers travaux concernant la parallélisation automatique ont commencé à paraitre au début des années 90. Ce sujet important est de plus en plus d'actualité, car il existe de nombreux programmes séquentiels facilement parallélisables.

Nous distinguons principalement deux types de modèles de parallélisation automatique : les modèles pour les architectures à mémoire distribuée et les modèles pour les architectures à mémoire partagée. Les modèles de parallélisation automatique pour les architectures à mémoire partagée sont les plus courants. Ils sont beaucoup plus faciles à développer et à mettre en place que les modèles destinés aux architectures à mémoire distribuée. Il existe par exemple quelques projets récents pour l'automatisation de la parallélisation de programmes séquentiels pour les architectures à mémoire partagée. Il s'agit, entre autres, du compilateur PLUTO (Bondhugula *et al.*, 2008) et d'une approche hybride basée sur MPI qui a été développée par Ashwin Kumar *et al.* (2006). PLUTO est un outil de parallélisation automatique pour des programmes séquentiels écrits en *C* à travers l'API OpenMP. L'approche hybride introduite par Ashwin Kumar *et al.* (2006) combine une analyse par blocs et une analyse fonctionnelle du code source du programme à paralléliser. Les segments du code source du programme à paralléliser sont associés à des blocs qui sont analysés pour déterminer s'ils contiennent des dépendances des données ou pas. Les boucles sont aussi associées à des blocs qui peuvent être parallélisés en utilisant la technique de parallélisation de boucles.

La parallélisation automatique est aussi une nouvelle fonctionnalité prise en charge par les compilateurs d'*Oracle Solaris Studio* (Oracle White Paper, 2010) pour les programmes écrits en *C/C++* ou en *Fortran*, grâce à l'ajout d'options telles que *-xautopar* ou *-xreduction* au niveau de la compilation. Par exemple, la première de ces deux options permet

à l'utilisateur de demander au compilateur de paralléliser tout le code source, alors que la seconde permet d'indiquer une série spécifique d'opérations (ou instructions) à paralléliser.

Il existe aussi quelques projets de grande envergure, datant des années 90, relatifs au développement de paralléliseurs automatiques ou de compilateurs parallèles pour des langages standardisés tels que *C/C++* et *Fortran* (Backus, 1978). Dans ce cadre nous pouvons citer :

- Le VFC (Vienna Fortran compiler) :

Le VFC (Benkner, 1999a) est un système de parallélisation de code source à code source. Le VFC reçoit un code source séquentiel et effectue les transformations pour générer un code source parallèle en langage HPF+ (Benkner, 1999b). HPF+ une version optimisée de HPF (High Performance Fortran) (Benkner et Zima, 1999). HPF+ offre à l'utilisateur la possibilité de spécifier certaines informations qui peuvent influencer grandement la performance générale du programme. Ces spécifications portent sur la localité des données, les accès aux données non-locales, etc. Les tests de performance ont démontré qu'un langage de programmation parallèle de haut niveau, tel HPF+, offre des performances proches de celles obtenues par les programmes parallèles d'échange de messages écrits à la main.

- Le compilateur *Polaris* (Polaris compiler) :

Le compilateur *Polaris* (Padua *et al.*, 1993) prend en entrée un programme écrit en *Fortran77* et effectue la transformation du programme pour qu'il fonctionne efficacement sur une machine parallèle. Le résultat fourni par ce compilateur est un programme parallèle dans une des versions parallèles de *Fortran*. Le format d'entrée inclut plusieurs directives qui permettent l'utilisation du parallélisme dans le programme source de manière explicite. La sortie du compilateur est habituellement au format *Fortran77*, enrichi de directives parallèles. Le compilateur s'acquitte des transformations en plusieurs passes de calculs ou de traitements (Padua *et al.*, 1993). En plus des nombreuses passes obligatoires, ce compilateur inclut des fonctionnalités avancées permettant l'exécution des tâches suivantes: la privatisation des tableaux, le

test des dépendances des données, la reconnaissance des variables d'induction, l'analyse inter-procédurale et l'analyse symbolique du programme.

- Le compilateur SUIF (SUIF compiler) :

Le compilateur SUIF (Stanford University Intermediate Format) (Hall *et al.*, 1996) prend *Fortran* et *C* comme langages d'entrée. Le code parallélisé repose sur la technique SPMD. Il existe deux principales étapes dans le processus de parallélisation automatique par le compilateur SUIF. Tout d'abord, l'étape d'analyse repère tout le parallélisme disponible dans le code. Cette étape englobe toutes les analyses et la détection de parallélisme entre les différentes procédures du programme. Le deuxième grand volet est l'optimisation et la génération du code parallèle. Les optimisations permettent d'améliorer les performances une fois que le parallélisme a été trouvé. En particulier, le compilateur SUIF complet incorpore des transformations des structures des données et des boucles pour augmenter la granularité du parallélisme, améliorer les accès à la mémoire des programmes et éliminer certaines synchronisations.

Dans le même cadre, nous pouvons citer aussi des projets tels que : *Fortran D* (Fox *et al.*, 1990), *Adaptor* (Brandes, 1993), projet *Esprit R&D Prepare* (Veen et de Lange, 1993), *EPPP* (Hurteau *et al.*, 1994) *Pandore* (André *et al.*, 1995) et le compilateur *Paradigm* (Banerjee *et al.*, 1995).

Il existe peu d'outils de parallélisation automatique pour les architectures à *mémoire distribuée*, disponibles gratuitement sur le Web. Un modèle théorique et pratique d'automatisation de parallélisation pour cette architecture a été introduit par Gasper *et al.* (2003). Selon ces derniers auteurs, ils ont implanté un système informatique permettant de transformer le code séquentiel en une structure représentant le flot d'exécution général du programme. Une représentation graphique de ce flot devrait aussi être disponible. Cependant, nous n'avons pas retrouvé sur le Web le produit final du projet décrit par Gasper *et al.* (2003).

Notons que l'utilisation des paralléliseurs automatiques ou compilateurs parallèles est assez limitée, car ces outils dépendent fortement de l'architecture de l'environnement de programmation parallèle mis en place.

4.6 Un nouvel outil de parallélisation semi-automatique

Il existe plusieurs raisons pour considérer la transformation d'un programme séquentiel en un programme parallèle. La première raison, qui est la plus souvent mentionnée, est l'existence de nombreux programmes séquentiels facilement parallélisables, dont le temps d'exécution est relativement long. Même si certaines applications ne peuvent pas être parallélisées entièrement et/ou automatiquement, les compilateurs parallèles et les automatiseurs de la parallélisation devraient faciliter la tâche du programmeur en ajoutant des instructions parallèles à certaines parties du code et en effectuant des transformations qui exploitent le parallélisme de bas niveau. Ces transformations, qui sont souvent très lourdes à faire manuellement, peuvent avoir une grande influence sur la performance globale d'un programme. La seconde raison est liée au fait que les puissants compilateurs parallèles devraient faciliter la programmation, en permettant ainsi le développement d'une grande partie du code dans des langages de programmation séquentiels familiers, tels que *C*, *Fortran*, etc. Bien sûr, les programmes parallèles développés doivent être portables entre différentes classes de machines. Le développement de programmes efficaces pour plusieurs machines parallèles est actuellement un grand défi, principalement à cause de la complexité architecturale de ces dernières. La grande variété de l'organisation des machines parallèles rend souvent la transformation d'un programme séquentiel existant en un programme parallèle plus difficile que sa reprogrammation complète pour une autre machine (Wilkinson et Allen, 2004). Une méthode de transformation automatique (ou semi-automatique) d'un programme séquentiel en un programme parallèle pourrait être vue comme un pont entre les langages de programmation séquentielle et les langages de programmation parallèle.

Nous avons développé un nouvel outil et une interface Web associée pour permettre la parallélisation de programmes bioinformatiques séquentiels écrits dans le langage *C*. La transformation complète et automatique d'un programme séquentiel en programme parallèle n'était pas vraiment envisageable dans le cadre de ce projet doctoral. Nous proposons ainsi

une transformation semi-automatique qui pourrait être vue comme un assistant au développeur de logiciels. L'outil que nous proposons sera basé sur les espaces de patrons de conception de programmes parallèles de MMS (voir la Figure 4.11). Nous avons identifié les principales étapes de cette transformation en utilisant la bibliothèque d'échange de messages MPI comme structure de soutien. Au niveau de toutes les étapes identifiées (voir la Figure 4.14), nous proposons une solution informatique et des conseils afin de simplifier le travail, diminuer le temps nécessaire au développement et augmenter la qualité du programme parallèle résultant. Pour guider le développeur à travers toutes les étapes de la conception, nous avons divisé le processus de parallélisation en cinq étapes principales qui sont les suivantes : la traduction des structures de données écrites en langage *C* en structures de données MPI, la détection des boucles et des dépendances des données associées, la génération du code parallèle, le partage et la synchronisation des données.

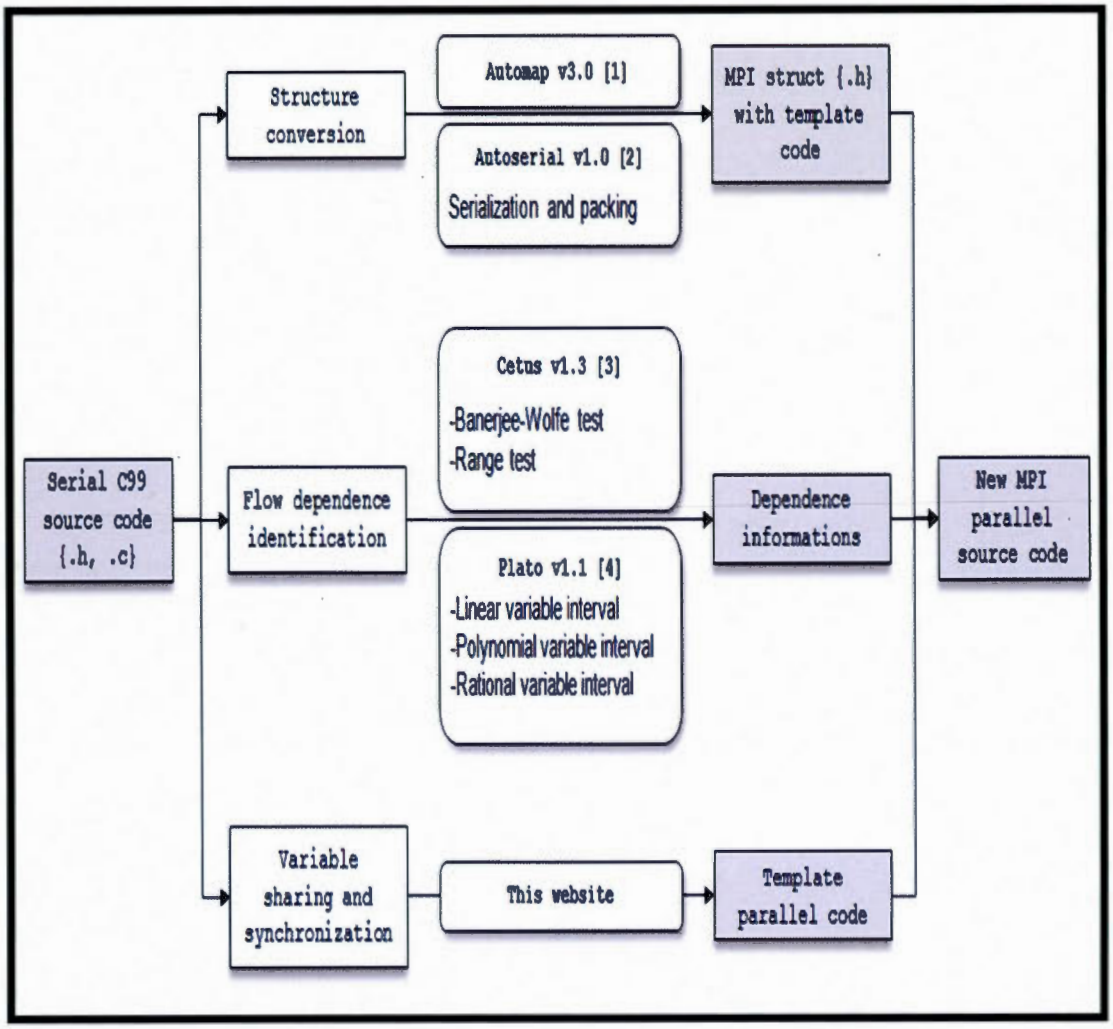


Figure 4.14 - Schéma du système de parallélisation semi-automatique que nous avons développé (figure tirée de Diallo *et al.*, 2012).

4.6.1 Étapes de traduction des structures de données

La première étape consiste à trouver une manière efficace et rapide pour traduire les structures de données, écrites en langage *C*, en structures de données utilisables par la bibliothèque d'échange de messages MPI.

Les structures de données utilisées par les programmes bioinformatiques sont souvent complexes et volumineuses. Une traduction manuelle de telles structures de données demeure

toujours possible. Cependant, une telle traduction peut générer des erreurs et nécessite souvent un temps non négligeable pour effectuer les tests afin d'assurer la validité de toutes les structures générées.

Il existe quelques outils permettant de transformer automatiquement des structures de données *C* en structures de données MPI. Parmi ces outils, nous avons : *OOMPI* (McCandless *et al.*, 1996), *Automap* (Devaney *et al.*, 1997), *C++2MPI* (Hillson et Iglewski, 2000), *TPO++* (Grundmann *et al.*, 2000), *MPIECC* (Renault, 2007), la bibliothèque *Autoserial* (Schaeli *et al.*, 2008), etc.

Nous avons automatisé cette étape à travers l'utilisation des deux outils suivants :

- *Automap* (Devaney *et al.*, 1997) : *Automap* est un outil qui facilite la création de structures de données MPI. C'est un compilateur qui traduit automatiquement des structures de données *C* en structures de données MPI. Pour plus de détails sur l'utilisation du compilateur *Automap*, le lecteur est référé à l'Annexe B.1.1.

- *Autoserial* (Schaeli *et al.*, 2008) : *Autoserial* est une bibliothèque *C++* permettant la déclaration d'objets sérialisables qui peuvent être facilement sauvegardés dans des fichiers ou envoyés sur le réseau. L'avantage majeur de cet outil est que le code de la sérialisation et de la désérialisation est généré automatiquement, ce qui rend la bibliothèque très facile à utiliser. Pour plus de détails sur l'utilisation de la bibliothèque *Autoserial*, le lecteur est référé à l'Annexe B.1.2.

Un script *Perl* (voir l'Annexe D.2) et un programme *Java* (voir l'Annexe D.3) ont été développés lors de la réalisation de cette étape. Le programme *Java* permet d'effectuer le processus de transformation des structures de données à travers *Automap* et *Autoserial*. Le script *Perl* développé permet d'exécuter *Automap* et d'afficher des messages d'erreurs simples, clairs et précis (voir l'Annexe D).

4.6.2 Détection des dépendances des données

L'automatisation de cette fonctionnalité a été réalisée en deux étapes :

Étape 1 **:** Nous avons utilisé tout d'abord le générateur de compilateurs et d'interpréteurs *SableCC* (Gagnon et Hendren, 1998). Écrit en langage *Java*, cet outil transforme la spécification grammaticale en un compilateur ou un interpréteur pour le langage décrit par cette spécification, évitant ainsi au programmeur la lourde tâche d'écrire le code des différentes parties d'un compilateur, telles que l'analyseur lexical ou l'analyseur syntaxique (Agbakpem, 2006). Pour plus de détails sur l'utilisation du compilateur *SableCC*, le lecteur est référé à l'Annexe B.3.

Pour pouvoir analyser les programmes à paralléliser à travers *SableCC*, la première étape consistait à écrire une grammaire pour le langage *C*. Nous avons écrit deux grammaires pour la partie de pré-compilation et une autre pour la partie compilation. Les grammaires de pré-compilation permettent d'ignorer toutes les lignes de codes liées à cette étape. Par la suite, nous avons écrit une grammaire (voir l'Annexe B.3) pour la partie de compilation en se basant sur la spécification décrite dans le document « *Committee Draft — ISO/IEC 9899:TC3* » et sur une grammaire développée par Roger Keays (Keays et Rakotonirainy, 2003). À travers ces deux grammaires, nous avons généré notre compilateur et nous l'avons intégré à un programme *Java* servant à traiter les boucles à paralléliser (voir l'Annexe E). Ce programme permet de détecter toutes les boucles du programme à paralléliser. Par la suite, nous identifions les informations importantes de chaque boucle, telles que les indices de boucle, le pas de progression, les variables modifiées, les variables utilisées dans les calculs, le niveau d'imbrication de la boucle, etc. Ces informations ont été détectées en utilisant les fonctions contenues dans les paquets générés par SableCC (voir l'Annexe B.3, pour plus de détails sur les paquets que nous avons générés, puis utilisés). Ces informations sont par la suite fournies aux outils d'analyses des dépendances des données (voir l'étape 2 ci-dessous).

Étape 2 **:** Une fois que nous avons détecté toutes les boucles contenues dans le programme, nous pouvons procéder à l'analyse des dépendances des données à travers la bibliothèque PLATO (Psarris et Kyriakopoulos, 2004; Kyriakopoulos, 2007) ou le compilateur *Cetus*

(Johnson *et al.*, 2005; Dave *et al.*, 2009). La bibliothèque PLATO (Programming Languages Analysis Translation and Optimization) contient toutes les méthodes d'analyse des dépendances développées par Psarris et son équipe (Psarris et Kyriakopoulos, 1999; Psarris et Kyriakopoulos, 2004; Kyriakopoulos, 2007). Le compilateur *Cetus* est une infrastructure *Java* qui permet une parallélisation automatique de programme séquentiels à travers OpenMP, en utilisant une la transformation de code *source à source*. Cette infrastructure inclut aussi deux méthodes de détection des dépendances des données : *Banerjee-Wolfe* (Wolfe et Banerjee, 1987) et *Range test* (Blume et Eigenmann, 1994). Pour plus de détails sur l'utilisation des outils PLATO et *Cetus*, le lecteur est référé à l'Annexe B.2.

4.6.3 Inclusion des instructions de parallélisation

Une fois que toutes les structures de données ont été traduites et que les tests d'analyse des dépendances des données ont été effectués, nous nous sommes intéressés à la génération du code parallèle. Cette étape consistait à inclure les commandes MPI pour diviser équitablement les charges de calculs effectués dans les boucles à paralléliser et à synchroniser les résultats obtenus. Tous les processeurs disponibles devaient effectuer approximativement le même nombre d'opérations de calculs en parallèle. Pour ce faire, nous suggérons l'utilisation de l'instruction de partage à travers l'operateur *modulo* :

SI (*indiceBoucle* % *nbProcesseurs* == *processeurId*);

où ***indiceBoucle*** représente l'indice de l'espèce (allant souvent de 0 au nombre d'espèces moins 1);

% représente l'opérateur *modulo*;

nbProcesseurs représente le nombre de processeurs utilisés lors de l'exécution du programme en parallèle;

processeurId représente l'identificateur du processeur courant.

La Figure 4.15 est un aperçu de l'utilisation de l'opérateur *modulo* pour la parallélisation des boucles imbriquées à travers MPI.

```
for (int i = 0 ; i < taille - 1 ;i ++ ){
   if (i % nbProcessors == processorID) {
      for ( int j = 0 ; j < taille - 1 ; j ++ ) {
         c [ i ] [ j ] = a [ i ] + b [ j ] ;
         }
      }
   }
```

Figure 4.15 - Exemple d'une boucle *for* imbriquée parallélisée à travers l'opérateur *modulo*.

Nous pouvons remarquer qu'avec l'inclusion de cette instruction, la boucle extérieure est distribuée équitablement entre les processeurs. Nous pouvons aussi utiliser d'autres formes de division des charges de travail pour les boucles à paralléliser. Par exemple, nous pouvons développer une fonction qui calcule le nombre de tours de la boucle et qui détermine le nombre de tours qui seront exécutés par chaque processeur (voir l'Annexe B.4).

4.6.4 Partage des données et synchronisation des résultats

Cette étape consiste à décomposer les données de façon équitable. Étant donné que la décomposition en fonction des données a été recommandée pour les programmes bioinformatiques que nous considérons (voir la section 4.4.2.1), nous pouvons appliquer un modèle de décomposition géométrique (voir la section 4.4.2.2). À l'aide du programme *Java* de détection des dépendances des données décrit dans la section 4.6.2, nous identifions toutes les boucles contenues dans le programme. Pour chaque boucle identifiée, nous détectons aussi les variables à distribuer entre les différents processeurs avant l'exécution de la boucle, de même que les variables à synchroniser sur les différents processeurs après l'exécution de la boucle. Pour identifier les variables à synchroniser, nous détectons toutes les variables modifiées lors de l'exécution de la boucle. Pour l'identification des variables à partager, nous détectons toutes les opérandes utilisées dans les instructions de la boucle. L'identification de ces variables a été effectuée à travers les fonctions contenues dans les paquets générés par SableCC. Ces deux listes de variables sont par suite suggérées à l'utilisateur afin de lui

faciliter la tâche de distribution et de synchronisation des données (voir l'Annexe E, pour plus de détails sur le programme *Java* développé).

Le partage et la synchronisation des données ont été exécutés en utilisant les routines de communication MPI (voir la section 4.2.4.2 pour plus de détails sur les routines de communication point à point et les routines de communication collectives).

Un pseudo-code pour le partage des données (voir l'Annexe F, Figures F.7 et F.8, pour un exemple concret) peut être défini comme suit :

Si (*processeurId* == *0*) alors

Envoi des données nécessaires aux autres processeurs (processeurID > 0).

À cette étape, nous pouvons par exemple utiliser la routine *MPI_Bcast()*.

Un pseudo-code pour la synchronisation des résultats (voir l'Annexe F, Figures F.7 et F.8, pour un exemple concret) peut être défini comme suit :

Si (*processeurId* > *0*) alors

Envoi de toutes les données nécessaires au premier processeur (*processeurID* =0).

À cette étape, nous pouvons par exemple utiliser la routine *MPI_Send()*.

Sinon

Réception des données envoyées par tous les processeurs à l'exception du premier (*processeurId* > 0).

À cette étape, nous pouvons par exemple utiliser la routine *MPI_Recv()*.

Traitement et gestion des résultats par le premier processeur.

Une interface Web permettant l'utilisation de tous les outils de parallélisation développés a été mise en place (voir l'Annexe F). Elle est disponible à l'adresse URL suivante : http://trex_cluster.labunix.uqam.ca.

4.7 Parallélisation du programme de détection de transferts horizontaux de gènes complets (*HGT-Detection*)

Lors de l'opération de profilage (voir le chapitre II, section 2.6), nous avons remarqué que les cinq fonctions suivantes : *copyInputTree*(*InputTree**, *InputTree*, *int*, *int*), *Floyd*(*double***, *double***, *int*, *int*), *approx_arb*(*double***, *double***, *double***, *double***, *int**, *long**, *double**, *int*, *int**, *int*, *int*), *Floyd*(*double***, *double***, *double***, *int*, *int*) et *BipartitionDistance*(*int***, *int***, *int*) totalisent 98% du temps total du programme (voir le Tableau 2.1). Nous avons aussi remarqué que dans la plupart des cas, ces cinq fonctions sont appelées à partir de la fonction *findBestHGTtab*(*InputTree, InputTree, Parameters, HGT*, int*, int*, int*) qui contient la boucle principale de la recherche des transferts horizontaux de gènes. La fonction *findBestHGTtab*(*InputTree, InputTree, Parameters, HGT*, int*, int*, int*), qui effectue le calcul des critères d'optimisation pour différentes paires de branches de l'arbre d'espèces donné, totalise presque 55% du temps total du programme (voir le Tableau 2.3).

Nous avons donc décidé de paralléliser la fonction *findBestHGTtab*, car nous l'avons identifiée comme étant la tâche principale de notre programme. Cette fonction correspond aux deux procédures entourées par le mot clé *Parallèle* dans la Figure 4.16 qui présente la version parallèle de l'algorithme *HGT-Detection*.

DÉBUT

Inférer les arbres d'espèces T et de gène T' sur le même ensemble d'espèces (i.e., feuilles);

Enraciner les arbres T et T' selon des évidences biologiques existantes ou en utilisant un *outgroup* ou un *point médian*;

Si (il existe des sous-arbres identiques, ayant au moins deux feuilles, entre T et T') alors

Diminuer la taille du problème en contractant les sous-arbres identiques dans T et T';

Sélectionner le critère d'optimisation *CO* entre : LS (least-squares), RF (distance de Robinson et Foulds), DQ (distance de quartets) et DB (dissimilarité de bipartitions);

Calculer la valeur initiale du critère d'optimisation *CO* entre T et T';

$T_0 = T$;

$k = 0$; //k est l'indice de l'étape

Tant que ($CO \neq 0$) {

(Parallèle)

Trouver l'ensemble des *THGs* candidats (*i.e.*, obtenus par des opérations SPR) dans l'arbre T_k, représenté par E_THG_k;

L'ensemble E_THG_k contient seulement les transferts satisfaisant la contrainte de sous-arbres;

(Parallèle)

Si (le rang du processeur = *0*) alors

Tant que (il existe des *THGs* satisfaisant les conditions des Théorèmes 1 et 2) {

Si (il existe *THGs* $\in E\_THG_k$ et satisfaisant les conditions du Théorème 2) alors

Effectuer les mouvements SPR correspondant à ces *THGs*;

Si (il existe *THGs* $\in E\_THG_k$ et satisfaisant les conditions du Théorème 1) alors

Effectuer les mouvements SPR correspondant à ces *THGs*;

}

Effectuer tous les mouvements SPR restants correspondant aux *THGs* satisfaisant la contrainte de sous-arbres;

Recalculer la valeur du critère d'optimisation *CO* après l'ajout de chaque *THG*;

$k = k + 1$;

Diminuer la taille du problème en contractant les sous-arbres identiques dans T_k et T';

Synchroniser les arbres T_k et T';

}

Éliminer les transferts inutiles obtenus;

FIN

Figure 4.16 - Algorithme parallèle du programme de détection de THGs complets, *HGT-Detection*. Les modifications par rapport à l'algorithme séquentiel correspondant sont indiquées en gras.

Notons qu'après l'exécution de la fonction parallèle de recherche des THGs, les transferts détectés sont stockés seulement dans le premier processeur, celui de *rang* 0 (voir la Figure 4.16). Les autres fonctions et instructions contenues dans la boucle *Tant que (CO ≠ 0)* de l'algorithme doivent aussi être exécutées par le processeur de *rang* 0. À la fin de chaque itération de la boucle *Tant que (CO ≠ 0)*, il est nécessaire de synchroniser les arbres d'espèces et de gène pour que tous les processeurs aient des données valides afin d'exécuter une fois de plus la version parallèle de la recherche des THGs si la condition de sortie de la boucle n'est pas satisfaite (voir la Figure 4.16).

Le pseudo-code détaillé de la fonction *findBestHGTtab* est présenté dans la Figure 4.17 et sa version parallèle est présentée dans la Figure 4.18.

```
DÉBUT
Calculer la valeur initiale du critère d'optimisation CO entre les arbres d'espèces T et de gène T';
Rechercher les bipartitions de T et de T';
Pour (i = 0; i < tailleArbreEspèces; i++) {
    Pour (j = 0; j < tailleArbresEspèces; j++) {
        Vérifier les hypothèses d'un transfert entre les espèces i et j;
        Si (le THG candidat satisfait la contrainte de sous-arbres) alors
            Ajouter le THG candidat à l'ensemble E_ THG;
    }
}
Retourner l'ensemble des transferts E_HGT;
FIN
```

Figure 4.17 - Pseudo-code de la version séquentielle de la fonction *findBestHGTtab*.

```
DÉBUT
Calculer la valeur initiale du critère d'optimisation CO entre les arbres d'espèces T et de gène T';
Rechercher les bipartitions de T et de T';
Pour (i = 0; i < tailleArbreEspèces; i++) {
    Si (i % nbProcesseurs = = processeurId ) alors
        Pour (j = 0; j < tailleArbresEspèces; j++) {
            Vérifier les hypothèses d'un transfert entre les espèces i et j;
            Si (le THG candidat satisfait la contrainte de sous-arbres) alors
                Ajouter le THG candidat à l'ensemble E_ THG;
        }
}
Envoyer tous les résultats au premier processeur (processeurId = 0);
Retourner l'ensemble des transferts E_HGT;
FIN
```

Figure 4.18 - Pseudo-code de la version parallèle de la fonction *findBestHGTtab*. Les instructions de parallélisation sont indiquées en gras.

Lors de la mise en œuvre de l'algorithme parallèle de recherche de THGs complets, nous avons effectué les étapes de transformation suivantes :

La transformation des structures de données : Nous avons tout d'abord transformé les structures de données *C* en structures de données MPI à travers *Automap* (voir l'Annexe B.1.1).

Le partage des données : Lors de cette étape, les arbres d'espèces et de gène sont distribués entre les processeurs. Le processeur ayant le *rang* zéro (processeur principal ou maître) transfère les données contenues dans les structures codant ses deux arbres aux autres processeurs, c'est-à-dire aux processeurs dont le *rang* se situe entre 1 et le nombre maximal de processeurs moins 1.

La parallélisation de la boucle principale de la recherche des transferts : L'analyse des dépendances des données de la boucle de la recherche des transferts (voir la Figure 4.18) a été effectuée à travers la librairie PLATO. Les résultats de cette analyse ont confirmé l'indépendance des données de la boucle. Nous l'avons parallélisée en utilisant l'instruction

modulo. Le test permettant d'évaluer l'hypothèse qu'un THG se soit produit entre une paire d'arêtes de l'arbre d'espèces, en se basant sur la topologie de l'arbre de gène, peut être réalisé en parallèle. La vérification de cette l'hypothèse, effectuée pour une paire d'arêtes, peut se faire indépendamment des autres paires. L'Annexe C.6 contient le code source de la version parallèle de la fonction de recherche de THGs complets (voir la fonction *findBestHGTtab*).

La synchronisation des résultats : Lors de cette étape, tous les processeurs effectuant la recherche des THGs complets transmettent au premier processeur (processeur de *rang* zéro) leurs résultats. Mentionnons que chaque processeur peut retrouver une partie des transferts détectés.

4.9 Performances du programme parallèle

Afin de mesurer les performances de notre programme parallèle, nous avons effectué plusieurs simulations. Ces simulations ont été réalisées en variant le nombre d'espèces impliquées, le nombre de transferts horizontaux détectés et le nombre de processeurs utilisés lors de l'exécution du programme parallèle. Ces simulations ont été réalisées sur le cluster *T-Rex* installé au laboratoire *LAMISS* de l'UQAM.

Le processus de simulations a été réalisé à travers les étapes suivantes :

Étape 1 : Génération des arbres phylogénétiques. Nous avons généré des arbres phylogéniques d'espèces aléatoires incluant : 20, 40, 80, 160 et 320 espèces (ou feuilles), respectivement. Nous avons ensuite introduit des transferts horizontaux aléatoires. Chaque arbre d'espèces contenait 10 ou 20 transferts qui le transformaient en arbre de gène. La procédure de Kuhner et Felsenstein (1994) que nous avons implémentée (voir l'Annexe C.5) a été utilisée pour générer les topologies d'arbres phylogénétiques d'espèces (voir aussi le chapitre II, la section 2.2.2, pour plus de détails sur cette procédure). Pour chaque combinaison de paramètres (nombre d'espèces, nombre de transferts), 100 paires d'arbres d'espèces et de gène aléatoires ont été générés, sauf pour le cas de 20 espèces et 20 transferts qui n'a pas été considéré dans nos simulations. Ce dernier cas n'est pas réaliste du point de vue biologique car nous ne pouvons pas avoir autant de transferts que de feuilles dans un arbre phylogénétique. Nous avons développé un script *Perl* et un programme *C++* pour la

génération des arbres phylogénétiques de différentes tailles contenant différents nombres de transferts (voir l'Annexe C.5).

Étape 2 : Exécution des programmes de détection de THGs avec les phylogénies générées. Lors de l'exécution des simulations pour le programme séquentiel, nous avons utilisé la version de base du programme *HGT-Detection* (évidement, ce programme ne contenait aucune instruction de parallélisation). Lors de l'exécution des simulations avec le programme parallèle, nous avons utilisé respectivement 2, 4, 8 et 12 processeurs afin d'estimer le temps d'exécution moyen du programme en fonction du nombre de processeurs. Le calcul du temps d'exécution du programme parallèle et du programme séquentiel a été réalisé à travers la routine *clock()* du langage *C*. Cette routine a été exécuté au début et à la fin de chacun des programmes.

Étape 3 : Calcul des performances. Nous nous sommes focalisés sur les gains au niveau du temps d'exécution, de l'accélération absolue et de l'efficacité du programme parallèle par rapport au programme séquentiel. Pour plus de détails sur les formules utilisées dans les calculs, voir la section 4.3 du chapitre.

Les résultats moyens obtenus lors des simulations (le temps moyen sur 100 jeux de données a été mesuré dans chaque cas) sont présentés dans la section 4.9.2.

4.9.1 Application de la loi d'Amdahl

Avant la présentation des résultats de nos simulations, nous avons considéré la loi d'Amdahl afin de calculer le gain de performance maximal que notre programme parallèle pourrait apporter. En utilisant la formule d'Amdahl (Amdahl, 1967) présentée dans la section 4.3.5, où N représente le nombre de processeurs et s représente la portion parallélisable du programme (comme nous l'avons établi dans chapitre II, section 2.6, la partie parallélisable du programme *HGT-Detection* est 55%), nous obtenons :

$$R = 20N / (9N + 11).$$

Alors le gain de performance maximal, *R*, du programme séquentiel de détection de THGs complets est égal à 1.38, 1.70, 1.93 et 2.02 pour 2, 4, 8 et 12 processeurs, respectivement.

4.9.2 Performances moyennes obtenues pour les cas de dix et vingt transferts horizontaux générés

Pour les arbres contenant 10 transferts horizontaux générés, nous remarquons que le programme parallèle devient plus rapide que le programme séquentiel à partir du cas de 40 espèces (voir la Figure 4.19). Pour le cas de 20 espèces, le temps d'initialisation de l'environnement parallèle, le temps des communications (transfert et synchronisation des données) entre les différents processeurs, le temps de terminaison de l'environnement, ainsi que le temps d'exécution des autres parties séquentielles du programme (environ 45% du temps total du programme) ne sont pas négligeables, même si le temps de recherche des THGs est amélioré en utilisant des instructions de parallélisation. Le temps total moyen de partage et de synchronisation des données, présenté sur la Figure 4.20, croit en fonction du nombre d'espèces considérées et du nombre de processeurs utilisés.

Le gain de temps obtenu par le programme parallèle, qui représente la différence entre le temps d'exécution du programme séquentiel et celui du programme parallèle, augmente au fur et à mesure que le nombre d'espèces augmente. Par exemple pour le cas 10 transferts et de 320 espèces, le temps d'exécution du programme séquentiel est d'environ 500 secondes, tandis que celui du programme parallèle, exécuté sur 12 processeurs, est d'environ 411 secondes, soit un gain de temps total de 89 secondes (voir la Figure 4.19). L'accélération maximale obtenue pour le cas de 10 transferts est de 1,39 avec 160 espèces et 12 processeurs utilisés (voir la Figure 4.21). De façon générale pour un nombre d'espèces fixe, à part le cas de 20 espèces, l'accélération absolue augmente avec le nombre de processeurs. L'accélération absolue varie en fonction du nombre d'espèces. Dans certains cas, elle commence à baisser à partir de 160 ou de 320 espèces, en fonction du nombre de processeurs utilisés. Cette baisse peut s'expliquer en partie par l'augmentation du temps de partage et de synchronisation des données. Une seconde hypothèse expliquant cette baisse de l'accélération absolue est basée sur l'exécution de la fonction *DeleteUseLessHGT(int, HGT*, InputTree,*

InputTree) qui permet de supprimer des transferts inutiles détectés et dont le temps d'exécution augmente proportionnellement à l'augmentation du nombre d'espèces. C'est la fonction qui consomme le plus de temps total moyen par appel est (voir le chapitre II, section 2.6).

Au niveau de l'efficacité (voir la Figure 4.22), nous remarquons que plus le nombre de processeurs augmente, plus l'efficacité baisse. Cette baisse de l'efficacité est due au fait que plus le nombre de processeurs augmente, moins les processeurs sont réellement utilisés (dans certaines situations, les processeurs peuvent ne pas être utilisés du tout). Notons que le phénomène de la baisse de l'efficacité des programmes parallèles, quand le nombre de processeurs augmente, est une caractéristique connue du processus de la parallélisation (pour plus de détails, voir le chapitre 12 dans Pacheco, 1996). Cette baisse peut aussi être expliquée par le fait que la courbe de l'accélération du programme parallèle a une croissance assez faible (voir la Figure 4.21).

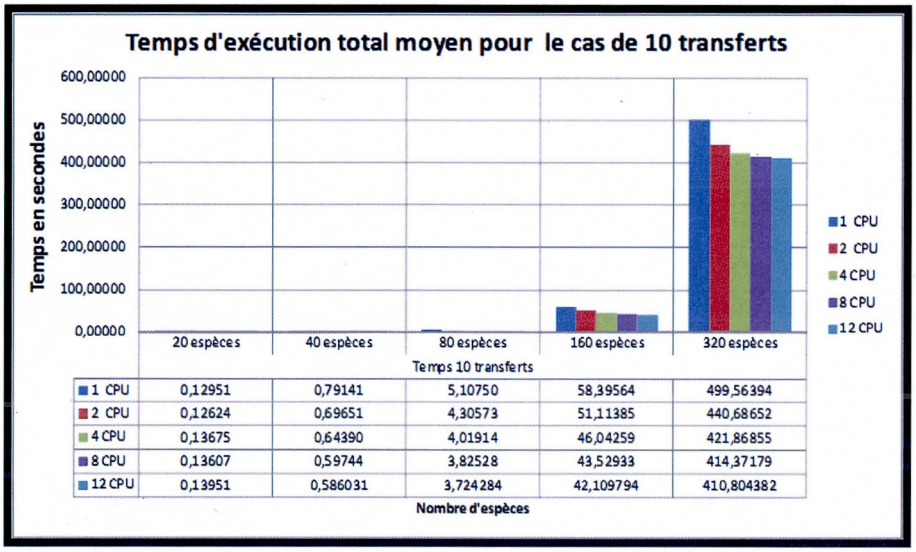
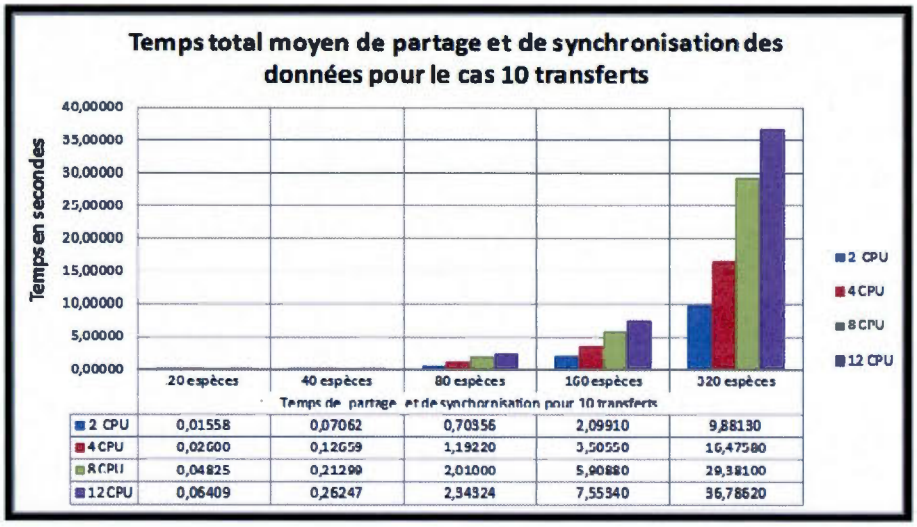
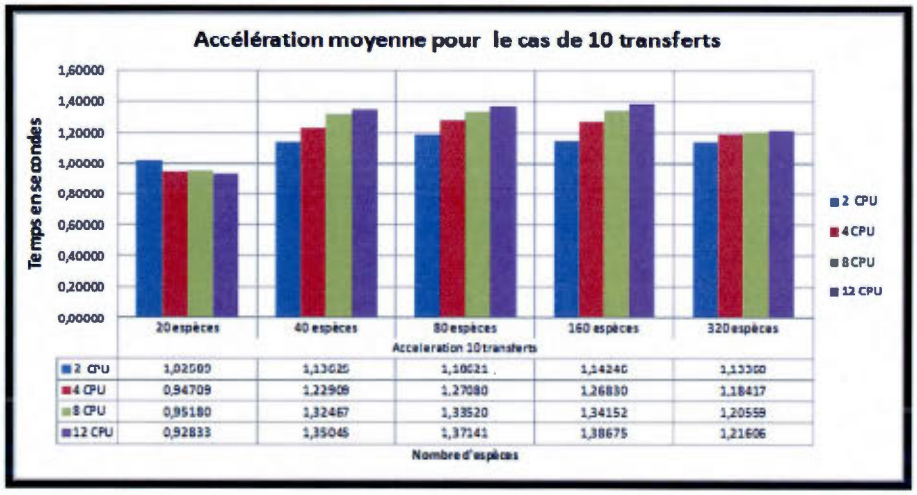


Figure 4.19 - Temps total moyen d'exécution pour le cas de 10 transferts. Le cas de 1 CPU représente le programme séquentiel.



Temps de partage et de synchronisation pour 10 transferts	20 espèces	40 espèces	80 espèces	160 espèces	320 espèces
2 CPU	0,01558	0,07062	0,70356	2,09910	9,88130
4 CPU	0,02600	0,12659	1,19220	3,50550	16,47580
8 CPU	0,04825	0,21299	2,01000	5,90880	29,38100
12 CPU	0,06409	0,26247	2,34324	7,55340	36,78620

Figure 4.20 - Temps total moyen de partage et de synchronisation des données pour le cas de 10 transferts.



Accelération 10 transferts	20 espèces	40 espèces	80 espèces	160 espèces	320 espèces
2 CPU	1,02569	1,13625	1,10621	1,14246	1,13360
4 CPU	0,94709	1,22908	1,27080	1,26830	1,18417
8 CPU	0,95180	1,32467	1,33520	1,34152	1,20559
12 CPU	0,92833	1,35045	1,37141	1,38675	1,21606

Figure 4.21 - Accélération absolue moyenne obtenue pour le cas de 10 transferts.

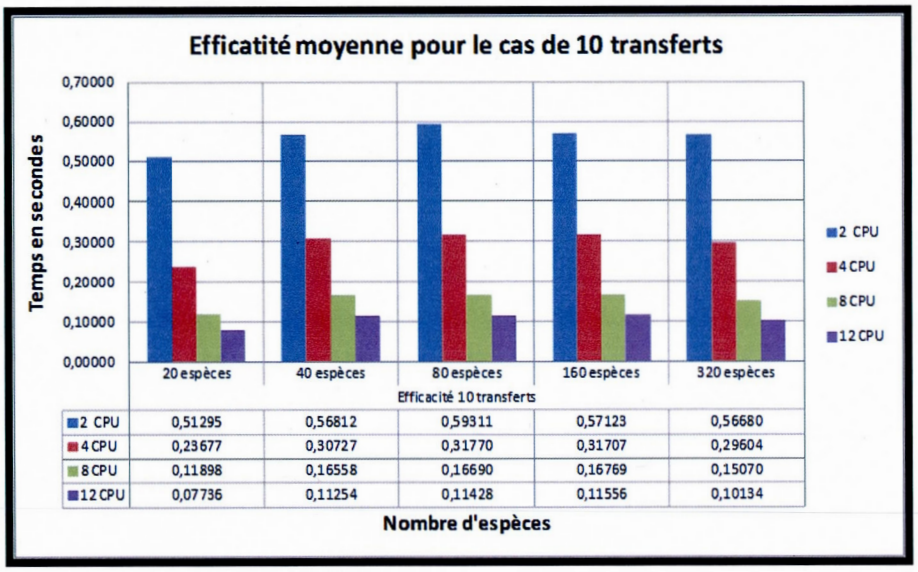


Figure 4.22 - Efficacité moyenne obtenue pour le cas de 10 transferts.

Pour les simulations impliquant 20 transferts générés (voir les Figures 4.23 - 4.26), nous pouvons faire les mêmes observations que dans le cas de 10 transferts générés. Pour le cas de 320 espèces, nous obtenons un gain de temps total de 252 secondes, sachant que le temps d'exécution du programme séquentiel est de 949 secondes, tandis que celui du programme parallèle, exécuté sur 12 processeurs, est de 697 secondes (voir la Figure 4.23). L'accélération maximale obtenue est de 1,45 avec 80 espèces et 12 processeurs utilisés (voir la Figure 4.25).

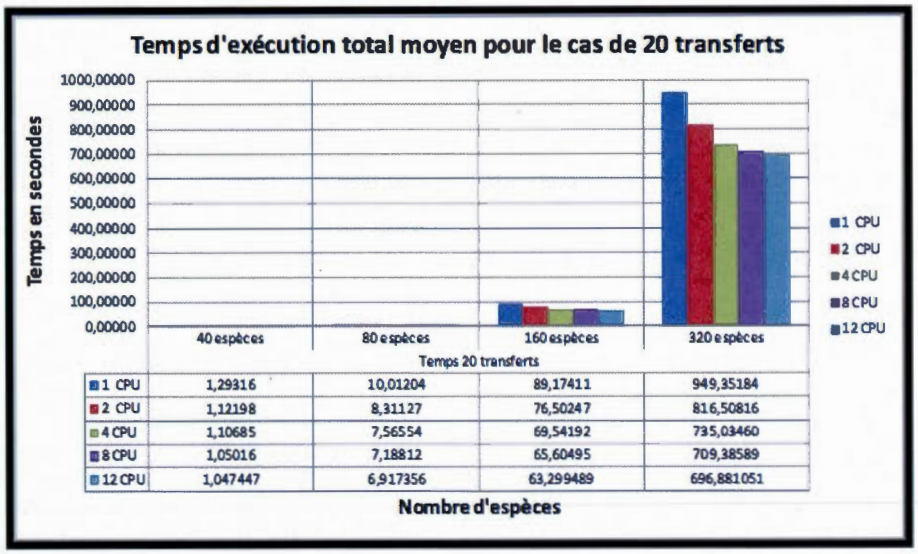


Figure 4.23 - Temps total moyen d'exécution pour le cas de 20 transferts. Le cas de 1 CPU représente le programme séquentiel.

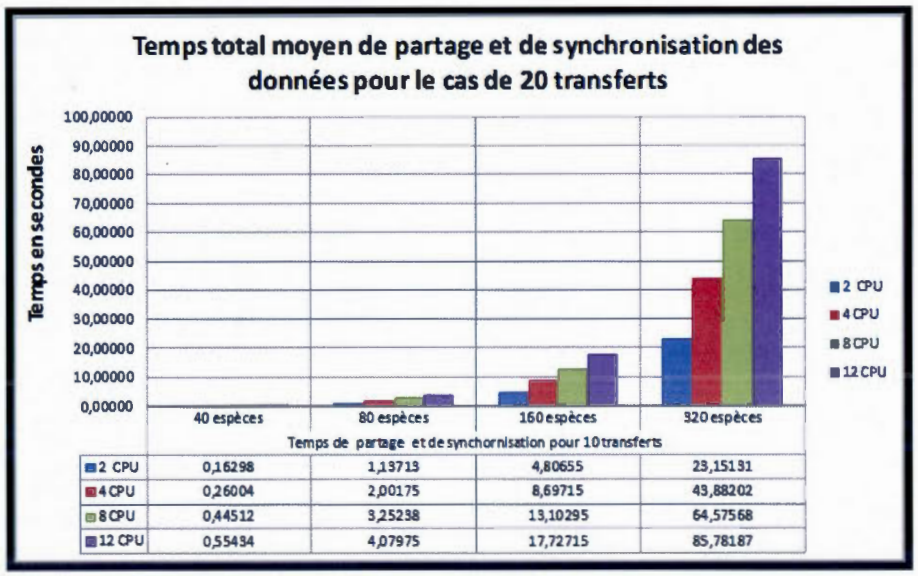


Figure 4.24 - Temps total moyen de partage et de synchronisation des données pour le cas de 20 transferts.

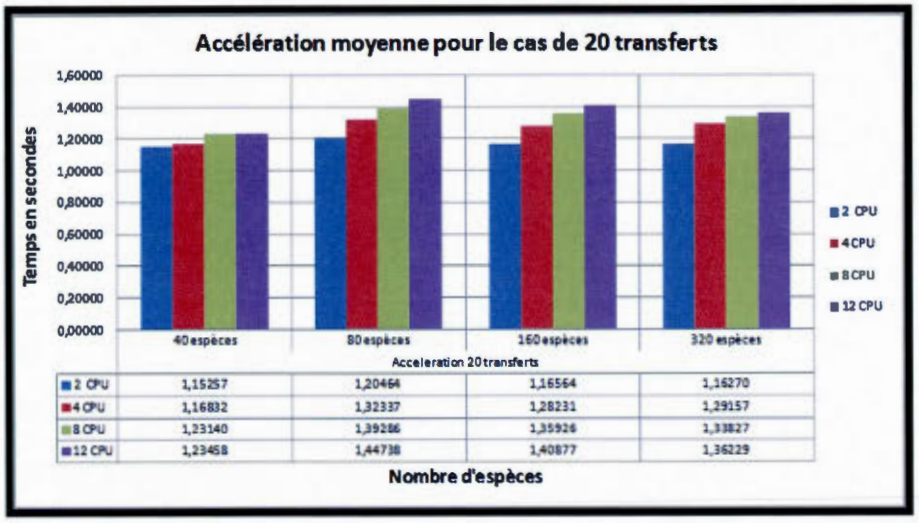


Figure 4.25 - Accélération absolue moyenne pour le cas de 20 transferts.

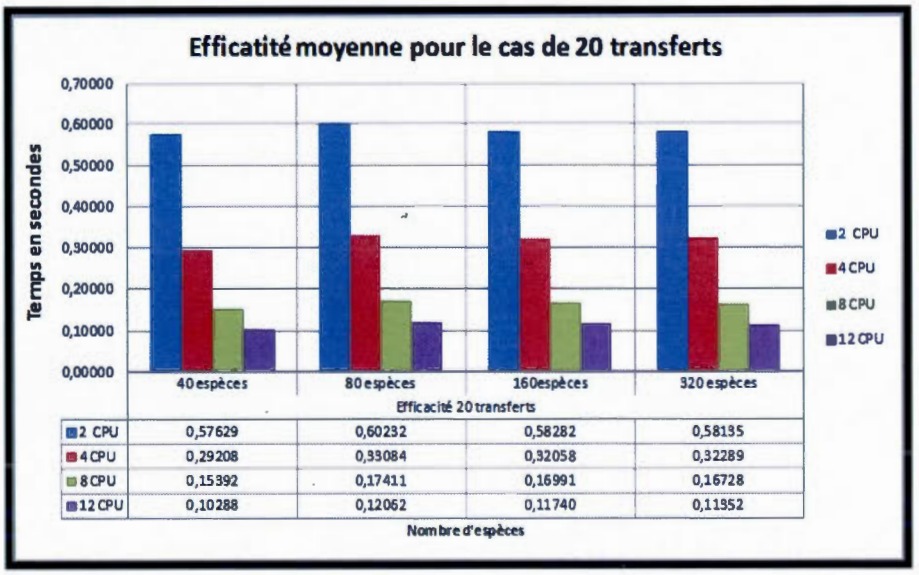


Figure 4.26 - Efficacité moyenne pour le cas de 20 transferts.

Pour résumer, nous pouvons observer que pour les deux cas étudiés, consistant en 10 et 20 transferts générés, l'algorithme parallèle fournit un gain de temps d'exécution à partir du cas de 40 espèces. Le gain de temps augmente au fur et à mesure que le nombre de transferts générés ou le nombre d'espèces considérées augmente. Ce gain de temps est particulièrement important au niveau de l'utilisation de l'interface Web du logiciel *T-Rex* (voir le chapitre V,

section 5.3). Lors des différentes simulations, nous avons aussi remarqué dans certains cas que le temps d'exécution du programme parallèle sur un arbre contenant x THGs pouvait être supérieur à celui contenant y transferts, où x était inférieur à y, pour des arbres ayant le même nombre d'espèces. Cette situation pourrait s'expliquer par le fait que la détection de THGs s'effectue dans une boucle et que les transferts détectés peuvent être indépendants ou dépendants les uns des autres. Par exemple, dans le cas de x transferts horizontaux, s'ils sont tous dépendants, la détection s'effectuera en x tours de boucle; c'est-à-dire un transfert détecté par tour de boucle. Alors que dans le cas de y transferts horizontaux indépendants, la détection s'effectuera en un seul tour de boucle, ce qui accélère le temps de calcul de façon significative.

Notons qu'à chaque nouvelle implémentation ou modification d'un programme, il est nécessaire d'effectuer un test de non régression. Pour ce faire, nous nous sommes servis de nos simulations pour déterminer si la version parallèle pouvait donner des résultats différents de ceux de la version séquentielle. La commande *cmp* de *Linux* a été utilisée pour la comparaison des fichiers des résultats (en format texte) fournis par les versions séquentielle et parallèle du programme. La seule différence entre les fichiers de sortie des programmes séquentiel et parallèle était l'ordre des transferts horizontaux de gènes détectés lors d'une même itération (dans nos simulations, cette différence apparaissait dans environ 12% des cas). Nous avons déterminé qu'une telle différence s'expliquait par le fait que ces transferts ont été détectés par des processeurs différents avant d'être envoyés au premier processeur (ce cas peut arriver seulement quand plusieurs transferts sont détectés lors d'une même itération). Mentionnons que cette différence dans l'ordre des transferts détectés n'est pas importante pour l'analyse et l'interprétation des résultats.

4.10 Résumé

Dans le chapitre IV, nous avons présenté les notions nécessaires à la compréhension de la programmation parallèle. Après avoir décrit quelques notions de base, nous avons exposé les principaux environnements de programmation parallèles existants, les espaces de conception de programmes parallèles de MMS, ainsi que quelques métriques de performance que nous avons utilisées dans ce projet doctoral. Nous avons aussi présenté un nouvel outil semi-

automatique de parallélisation d'algorithmes séquentiels (Diallo *et al.* 2012), incluant plusieurs étapes nécessaires à la transformation d'un programme séquentiel en programme parallèle. Finalement, nous avons décrit la version parallèle de l'algorithme de détection de THGs complets et nous avons exposé les résultats des tests de performance des programmes séquentiel et parallèle effectués sur différents jeux de données.

CHAPITRE V

INTERFACES WEB DES PROGRAMMES DE DÉTECTION DE TRANSFERTS HORIZONTAUX DE GÈNES

Dans ce chapitre, nous présentons les interfaces Web des différents programmes de détection de transferts horizontaux de gènes complets et partiels. Les interfaces suivantes seront présentées - (1) celle de la version séquentielle de la détection de THGs complets avec une option de bootstrap, (2) celle de la version parallèle de la détection de THGs complets, (3) celle de la version interactive de la détection de THGs complets, (4) celle la version consensus de la détection de THGs complets, (5) ainsi que celle de la version séquentielle de la détection de THGs partiels avec une option de bootstrap. Ces interfaces ont été ajoutées à une plateforme d'analyse phylogénétique : la version Web du logiciel *T-Rex* (Tree and reticulogram REConstruction).

La version Windows de *T-Rex* a été initialement développée et décrite en 2001 (Makarenkov, 2001). Elle incluait de nombreuses applications et algorithmes d'analyse phylogénétique. Nous avons mis en place la version Web de *T-Rex* en utilisant le langage de balisage *HTML* (Powell *et al.*, 1999) et le langage *PHP* (Hughes et Zmievski, 2001) pour le développement de l'interface et la gestion des données provenant des formulaires. Des scripts écrits en langage *Perl* ont été utilisés pour encapsuler l'exécution des programmes externes souvent écrits dans les langages de programmation *C* et *C++*. L'adresse URL (Uniform Resource Locator) de notre interface est la suivante : http://www.trex.uqam.ca.

5.1 Jeu de données considéré

Pour présenter les différentes interfaces des programmes de détections de transferts horizontaux de gènes que nous avons développées, nous utiliserons un jeu de données réelles traitant de l'évolution du gène *rpl12e*, originalement étudié par Matte-Tailliez *et al.* (2002). Matte-Tailliez *et al.* (2002) ont inféré un arbre de maximum de vraisemblance (ML) du gène *rpl12e* (voir la Figure 5.1) pour 14 organismes d'archées et l'ont comparé à la phylogénie ML (voir la Figure 5.2) basée sur la concaténation de 53 protéines ribosomales (7,175 positions). Des problèmes rencontrés lors de la reconstruction de certaines parties de la phylogénie des archées ont permis d'émettre une hypothèse selon laquelle des transferts horizontaux ont grandement influencé l'évolution du gène *rpl12e*.

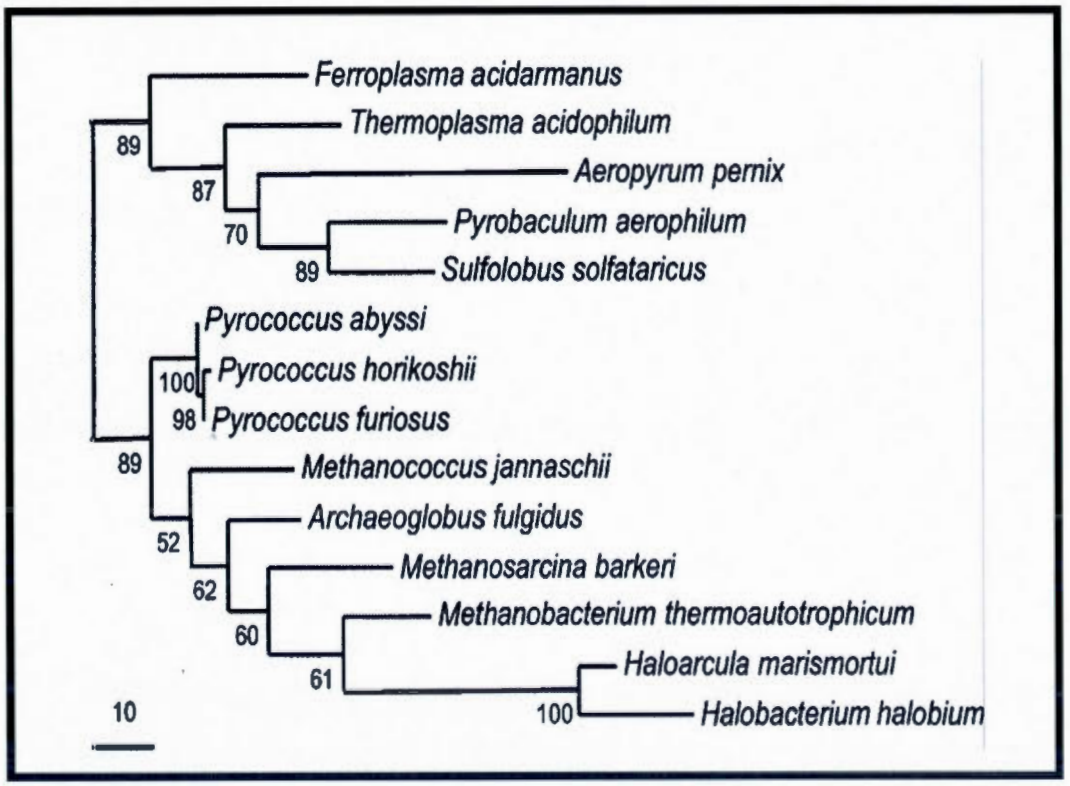


Figure 5.1 - Arbre de maximum de vraisemblance du gène *rpl12e* produit par Matte-Tailliez *et al.* (2002). Les scores de bootstrap associés aux branches de l'arbre sont indiqués.

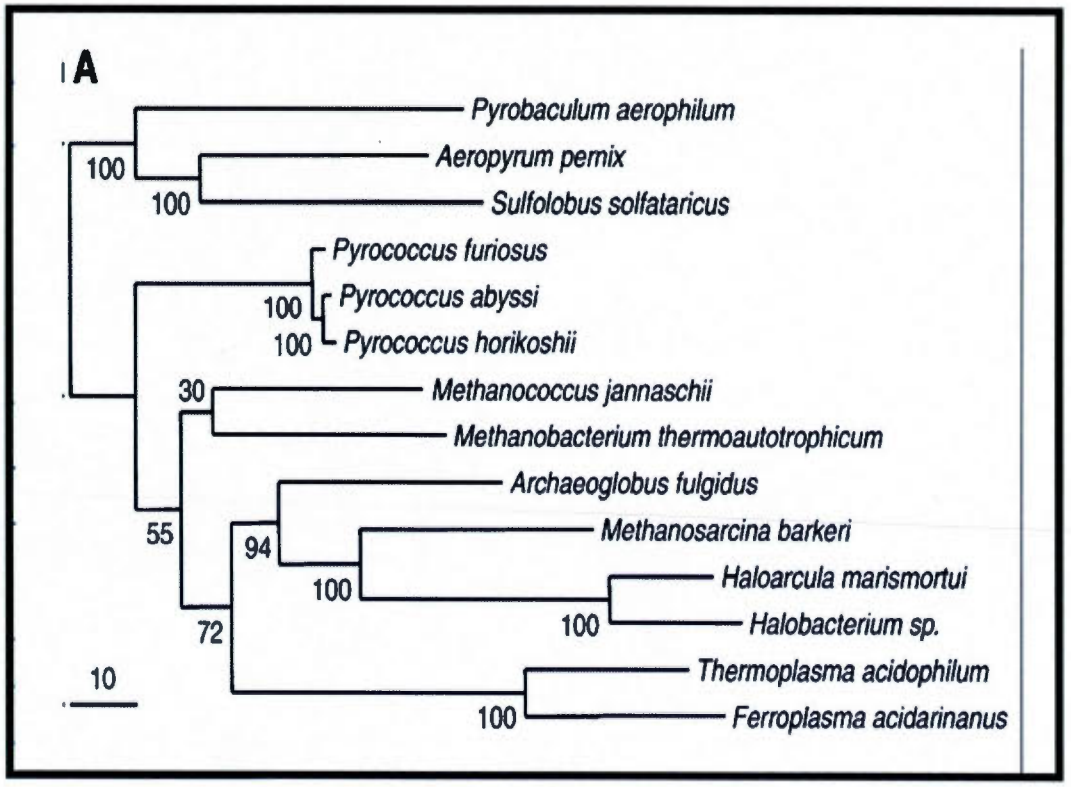


Figure 5.2 - Arbre de maximum de vraisemblance basé sur la concaténation de 53 protéines ribosomales (7,175 positions) produit par Matte-Tailliez *et al.* (2002). Les scores de bootstrap associés aux branches de l'arbre sont indiqués.

Vu l'incongruence topologique des phylogénies obtenues, les auteurs ont prédit quelques cas de transferts du gène *rpl12e*. Plus précisément, le cas du transfert entre les clades des organismes Thermoplasmatales (*Ferroplasma acidarmanus* et *Thermoplasma acidophilum*) et Crenarchaeota (*Aeropyrum pernix*, *Pyrobaculum aerophilum* et *Sulfolobus solfataricus*) a été indiqué comme le plus évident.

Dans cette thèse, nous avons considéré les topologies des arbres de gène (voir la Figure 5.3) et d'espèces (voir l'arbre de base sur la Figure 5.4) pour le même jeu de données, reconstruites dans Boc *et al.* (2010). La Figure 5.4 représente les transferts obtenus en utilisant l'algorithme *HGT-Detection* (Boc *et al.*, 2010), présenté dans le chapitre II.

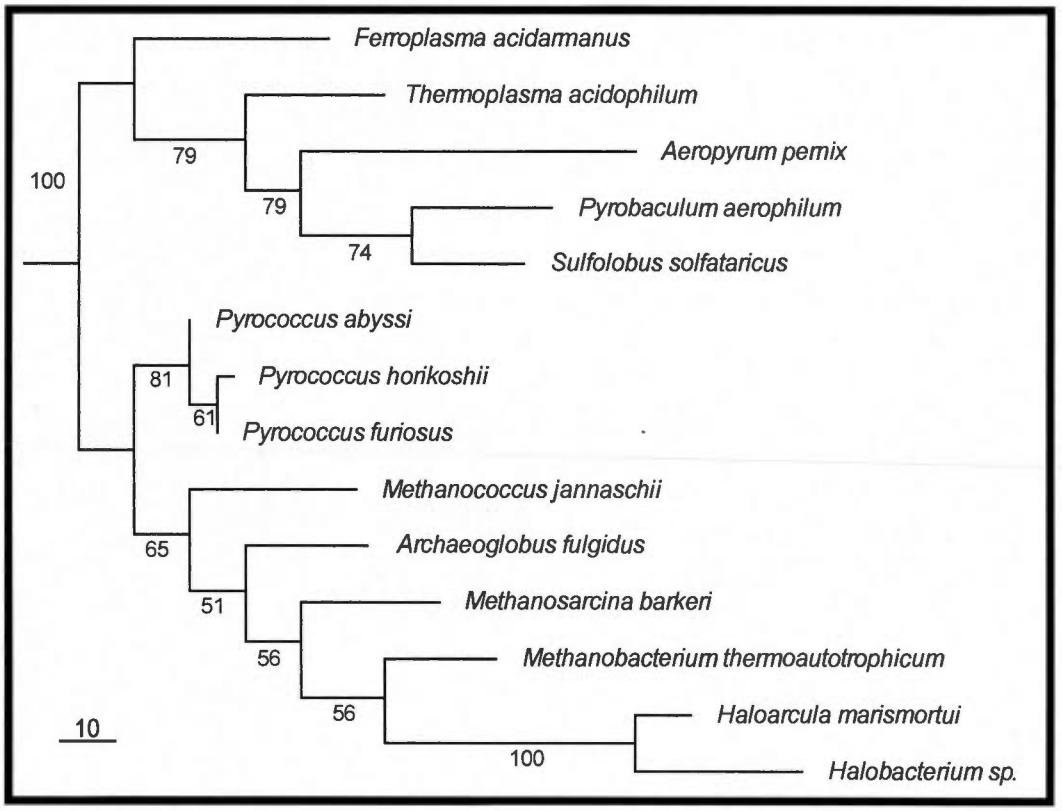


Figure 5.3 - Arbre de maximum de vraisemblance du gène *rpl12e* reconstruit par Boc *et al.* (2010). Les scores de bootstrap associés aux branches de l'arbre, obtenus en utilisant les programmes *Seqboot* et *Protml* (modèle JTT, Jones *et al.*, 1992) du logiciel *PHYLIP* (Felsenstein, 1989), sont indiqués.

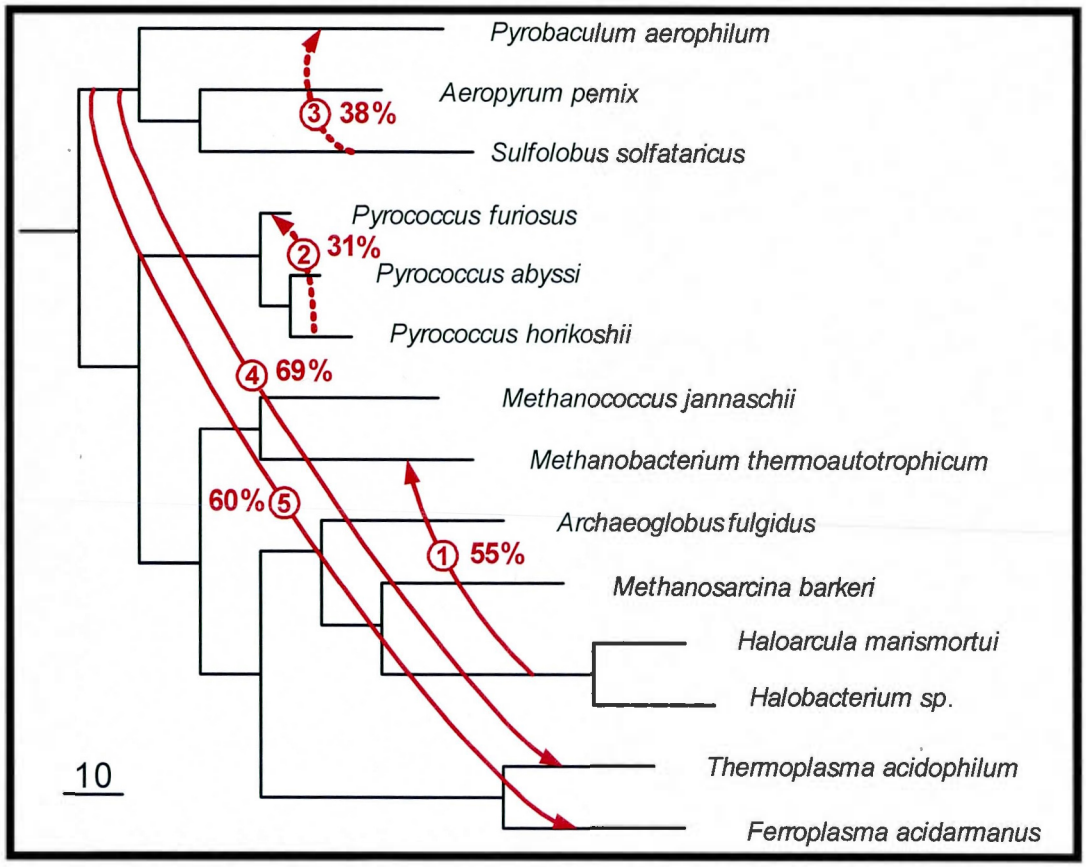


Figure 5.4 - Scénario de transferts obtenu par l'algorithme *HGT-Detection* (Boc *et al.*, 2010) appliqué au jeu de données du gène *rpl12e*. Le score de bootstrap des THGs est indiqué à côté de chaque transfert détecté. Les flèches 4 et 5 illustrent les transferts entre les clades des Thermoplasmatales et des Crenarchaeota prédits par Matte-Tailliez *et al.* (2002). Les transferts avec un score de bootstrap de 50% ou moins sont illustrés par des flèches en pointillé.

Cinq transferts nécessaires pour réconcilier les arbres d'espèces et de gène ont été trouvés par *HGT-Detection* (ils sont indiqués par des flèches sur la Figure 5.4). Le transfert entre le clade (*Halobacterium sp.* et *Haloarcula marismortui*) et l'organisme *Methanobacterium thermoautotrophicum* a été trouvé à la première étape. Son support de bootstrap, calculé en fixant la topologie de l'arbre d'espèces et en répliquant les séquences de l'arbre de gène, est de 55%.

À la deuxième et à la troisième étape, les transferts entre *Pyrococcus horikoshii* et *Pyrococcus furiosus* (étape 2), et entre *Sulfolobus solfataricus* et *Pyrobaculum aerophilum* (étape 3), ont été détectés. Ces deux THGs lient des espèces proches et ont donc de faibles scores de bootstrap.

Les transferts 4 et 5 lient le clade des Crenarchaeota aux organismes *Thermoplasma acidophilum* et *Ferroplasma acidarmanus*. Les transferts entre ces deux groupes ont été prédits par Matte-Tailliez *et al.* (2002). La direction identique et les scores de bootstrap similaires des THGs 4 et 5 suggèrent qu'un *unique transfert horizontal*, au lieu des deux transferts indiqués, pourrait avoir lieu entre les clades de Thermoplasmatales et Crenarchaeota. Il est à noter que tout algorithme standard basé sur la minimisation de la distance SPR (Subtree Prune and Regraft) devrait trouver deux transferts dans ce cas. Un transfert unique reliant ces clades serait vraisemblablement caché suite à un artéfact de reconstruction de l'arbre de gène. L'interface interactive, présentée dans la section 5.4, que nous avons développée permettra de retrouver ce transfert unique à la place des deux transferts identifiés par l'algorithme *HGT-Detection* standard.

5.2 Interface du *programme séquentiel* de détection de transferts horizontaux de gènes complets

Le programme séquentiel de détection de THGs complets, *HGT-Detection*, procède par une réconciliation graduelle d'une phylogénie enracinée d'espèces et d'une phylogénie enracinée d'un gène (Makarenkov *et al.*, 2007 et Boc *et al.*, 2010). À chaque étape du processus de réconciliation ou de détection, plusieurs paires d'arêtes de l'arbre d'espèces sont évaluées selon l'hypothèse qu'un THG se soit produit entre elles et les transferts horizontaux détectés sont ajoutés progressivement à l'arbre phylogénétique d'espèces. Pour plus d'informations, le lecteur est référé au chapitre II (section 2.4).

L'interface du programme séquentiel de détection de THGs complets (voir la Figure 5.5) permet à l'utilisateur de spécifier l'arbre d'espèces et l'arbre de gène (codés en format Newick) qui seront utilisés lors du processus de réconciliation. Cette interface inclut les options suivantes :

- la validation des transferts à travers le bootstrap. Dans le cas d'une validation par bootstrap, l'utilisateur doit spécifier plusieurs arbres de gène;
- la sélection de la racine de l'arbre d'espèces et de l'arbre de gène;
- le mode de détection de transferts horizontaux (un seul transfert détecté par itération ou plusieurs transferts détectés par itération);
- la sélection du critère d'optimisation : le critère des moindres carrés, la distance topologique de Robinson et Foulds ou la dissimilarité de bipartitions.

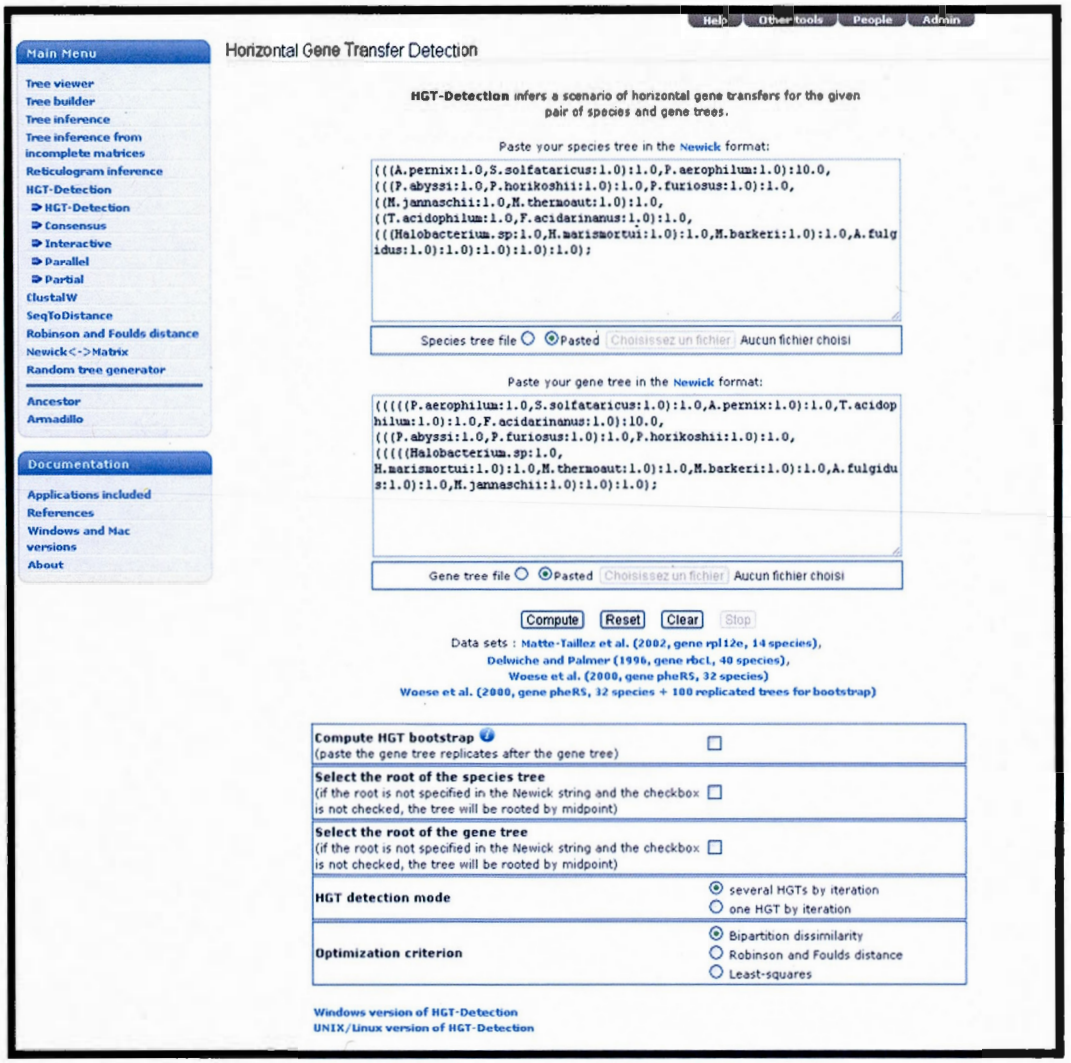


Figure 5.5 - Interface Web du programme séquentiel de détection de THGs complets.

Les cinq transferts (décrits dans la section 5.1) nécessaires pour réconcilier les topologies d'espèces et de gène ont été trouvés pour les données du gène *rpl12e* Matte-Tailliez *et al.* (2002) (voir la Figure 5.6). La méthode *HGT-Detection* (Boc *et al.*, 2010) standard a été appliquée pour obtenir ces résultats. Un fichier texte présentant les résultats détaillés de la détection est aussi fourni par le programme. Le réseau de THGs obtenu peut être sauvegardé en format SVG (Scalable Vector Graphics) et puis modifié dans l'éditeur d'image de choix de l'utilisateur. L'arbre de base peut être présenté dans les formats hiérarchique vertical, hiérarchique horizontal, axial et radial. Les couleurs des branches de

l'arbre, des taxa et des réticulations (transferts détectés) peuvent être spécifiées par l'utilisateur. La racine de l'arbre présenté peut aussi être changée.

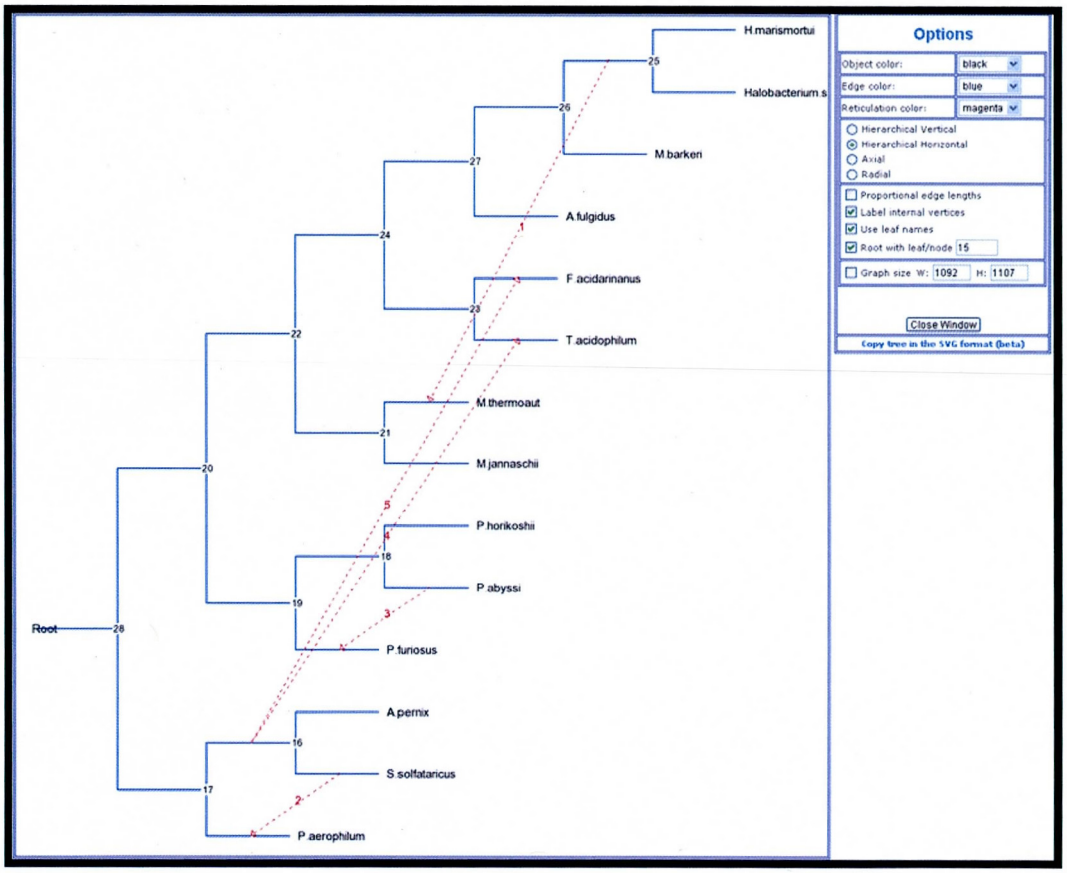


Figure 5.6 - Visualisation des résultats du programme séquentiel de détection de THGs complets.

5.3 Interface du *programme parallèle* de détection de transferts horizontaux de gènes complets

Le programme parallèle de détection de THGs complets résulte de la parallélisation du programme séquentiel *HGT-Detection*. Pour plus d'informations, le lecteur est référé au chapitre IV (section 4.7). Cette interface Web présentée sur la Figure 5.7 permet à l'utilisateur d'accélérer de façon significative la vitesse de détection de THGs complets retrouvés par l'algorithme *HGT-Detection*. L'utilisateur doit spécifier l'arbre d'espèces et

l'arbre de gène (codés en format Newick) qui seront utilisés lors du processus de détection. Cette interface contient les options suivantes :

- la sélection de la racine de l'arbre d'espèces et de l'arbre de gène;
- la sélection du critère d'optimisation : le critère des moindres carrés, la distance topologique de Robinson et Foulds ou la dissimilarité de bipartitions.

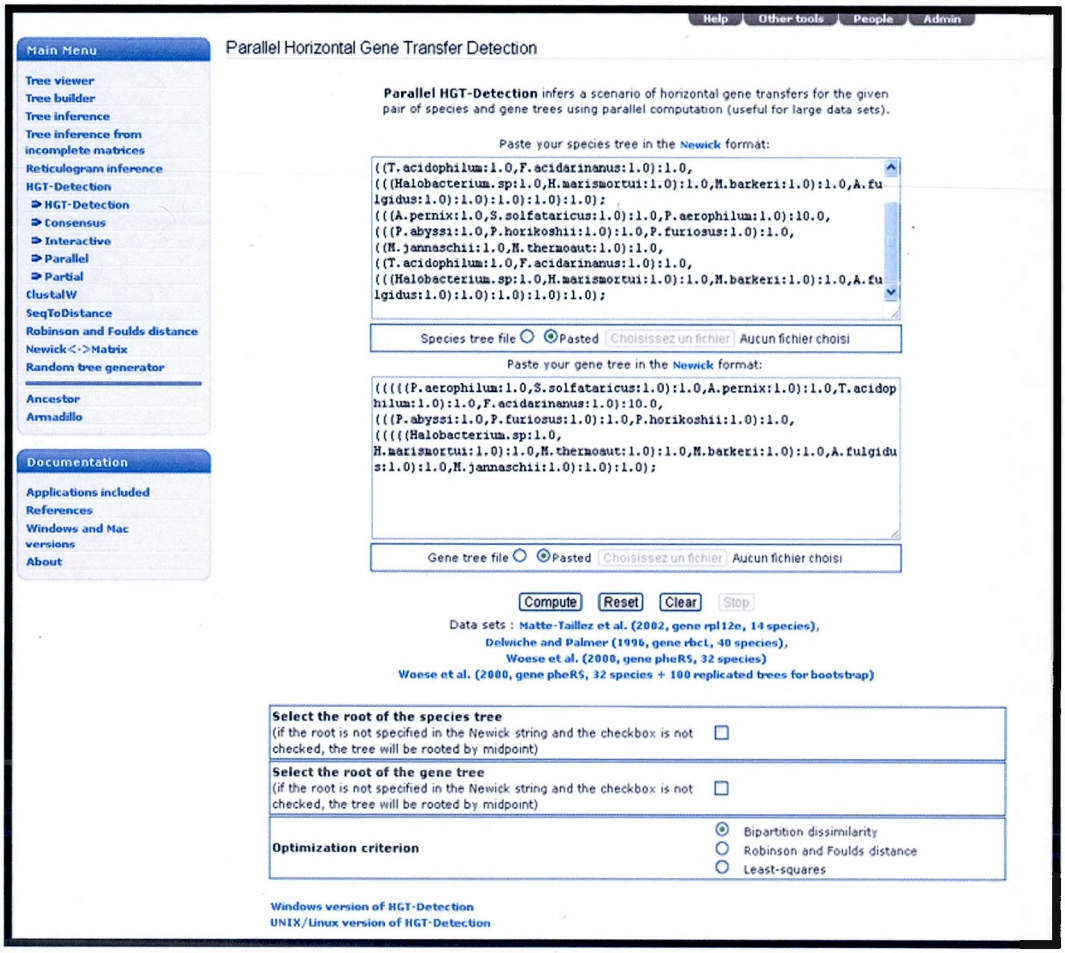


Figure 5.7 - Interface Web du programme parallèle de détection de THGs complets.

Les mêmes statistiques de détection, ainsi que les mêmes options de manipulation des résultats graphiques, que celles décrites dans la section 5.3, sont aussi disponibles à la suite de l'exécution de ce programme parallèle. Pour l'exemple présenté dans la section 5.1, les mêmes cinq transferts horizontaux sont détectés par la version parallèle du programme *HGT-Detection* que par sa version séquentielle (voir la Figure 5.6).

5.4 Interface de la *version interactive* du programme de détection de transferts horizontaux de gènes complets

Étant donné que la détection de transferts horizontaux de gènes se fait automatiquement par la version standard de *HGT-Detection*, sans connaissances *a priori* sur les données biologiques traitées, nous avons mis en place un processus de validation des transferts détectés afin d'augmenter la plausibilité des résultats du programme. À travers l'utilisation de la *version interactive*, un biologiste ou un bioinformaticien, disposant d'informations biologiques complémentaires sur les arbres traités, aura un meilleur contrôle sur les transferts détectés. Ce contrôle inclut le processus de validation, mais aussi une option d'ajout de transferts initiaux. Les transferts initiaux suggérés par l'utilisateur sont testés et considérés lors de la détection de transferts suivants. Pour plus de détails, le lecteur est référé au chapitre II (section 2.5).

L'interface interactive (voir la Figure 5.8) demande à l'utilisateur de spécifier tout d'abord l'arbre d'espèces et l'arbre de gène (codés en format Newick). Les options suivantes sont incluses dans cette interface :

- la suggestion de transferts initiaux;
- la sélection de la racine de l'arbre d'espèces et de l'arbre de gène;
- le mode de détection de transferts horizontaux (un seul transfert détecté par itération ou plusieurs transferts détectés par itération);
- la sélection du critère d'optimisation : le critère des moindres carrés, la distance topologique de Robinson et Foulds ou la dissimilarité de bipartitions.

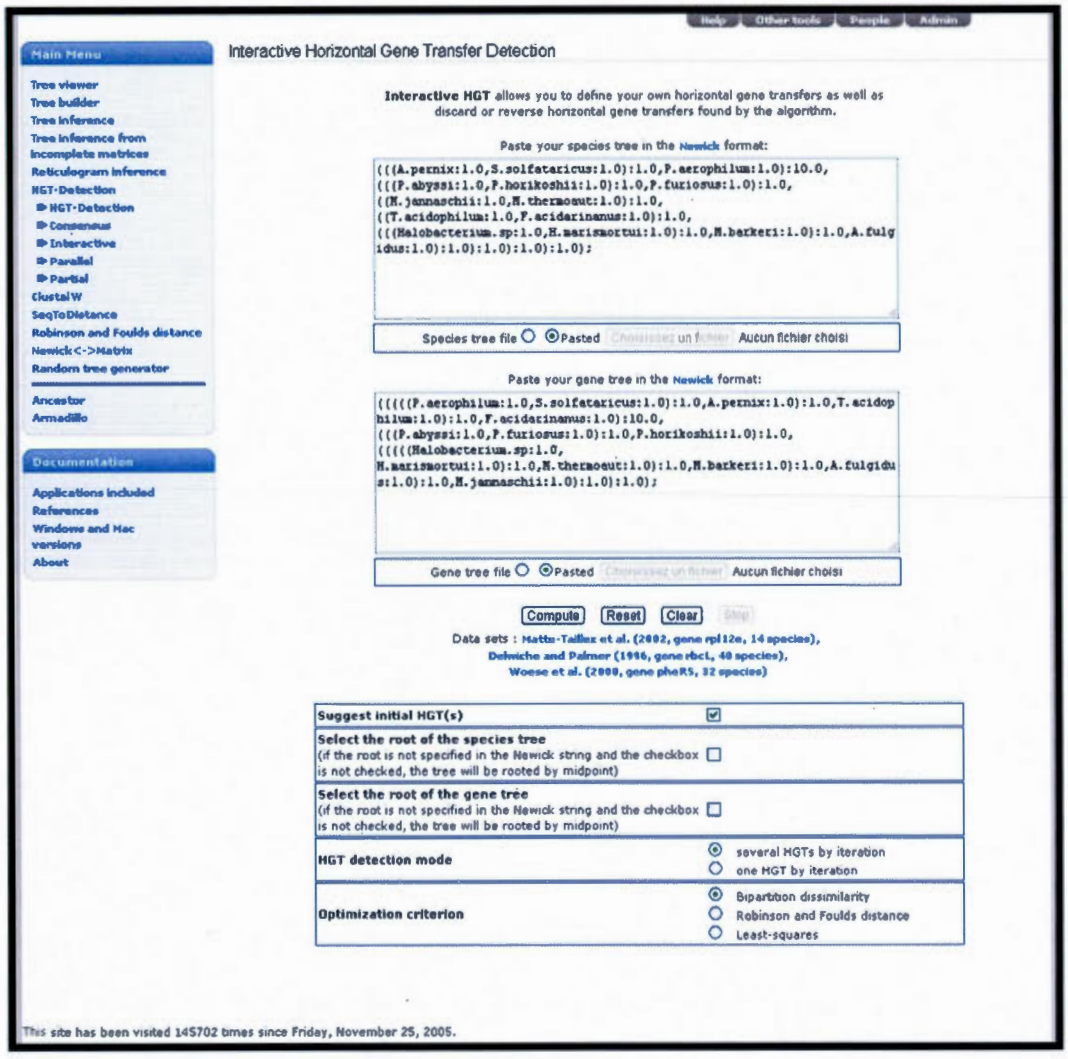


Figure 5.8 - Interface Web de la version interactive du programme de détection de THGs complets.

En utilisant l'interface de la version interactive, nous pouvons sélectionner des transferts initiaux (voir la Figure 5.9) : par exemple, les transferts 4 et 5 qui lient le clade des Crenarchaeota aux organismes *Thermoplasma acidophilum* et *Ferroplasma acidarmanus* (voir l'exemple de données considéré dans la section 5.1) peuvent être représentés par un transfert unique entre ces deux clades, qui est suggéré ci-haut. À l'étape initiale, nous proposons donc un transfert entre les clades de Thermoplasmatales et Crenarchaeota (HGT_1 entre les branches 16-17 et 23-24 de l'arbre sur la Figure 5.9).

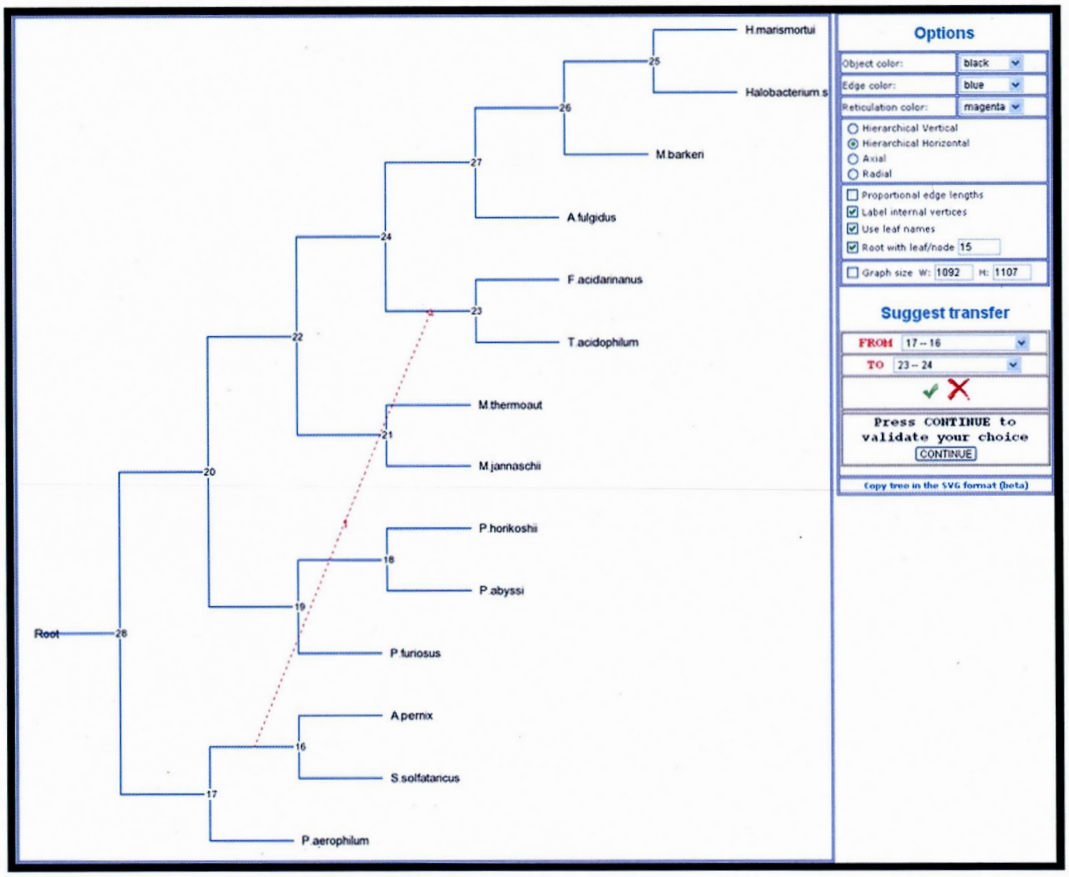


Figure 5.9 - Sélection d'un transfert initial (HGT_1) entre les clades de *Thermoplasmatales* et *Crenarchaeota*.

Puis, à la première itération de la version interactive de *HGT-Detection*, trois transferts ont été détectés (voir la Figure 5.10) : le transfert entre le clade (*Halobacterium sp.* et *Haloarcula marismortui*) et l'organisme *Methanobacterium thermoautotrophicum* (HGT_2 sur la Figure 5.10), le transfert entre les organismes *Sulfolobus solfataricus* et *Pyrobaculum aerophilum* (HGT_3 sur la Figure 5.10) et le transfert entre les organismes *Pyrococcus abyssi* et *Pyrococcus furiosus* (HGT_4 sur la Figure 5.10).

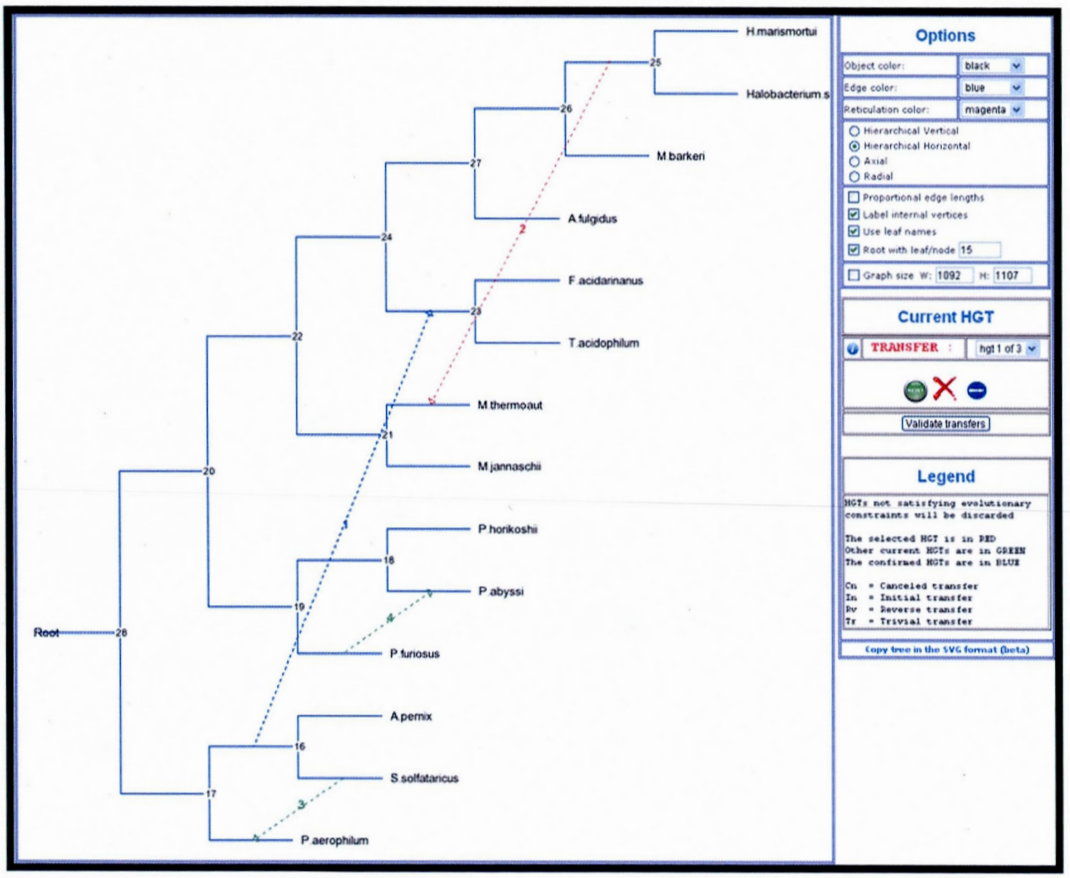


Figure 5.10 - Transferts (HGT_2, HGT_3 et HGT_4) détectés à la première itération de la version interactive du programme de détection de THGs complets.

À la seconde itération (voir la Figure 5.11), nous avons trouvé le transfert entre le clade des *Crenarchaeota* et l'organisme *Thermoplasma acidophilum* (HGT_5 sur la Figure 5.11).

À travers le processus de gestion mis en place, ce dernier transfert doit être annulé car le transfert initial (HGT_1) serait suffisant pour expliquer la proximité des clades en question (voir la Figures 5.9). Au total, un scénario de THGs de coût minimal, de quatre transferts, a été trouvé et validé à travers notre version interactive du programme, pour les arbres d'espèces et de gène considérés (voir la Figure 5.12). Cette solution est donc plus parcimonieuse que la solution fournie par la version standard de l'algorithme *HGT-Detection*.

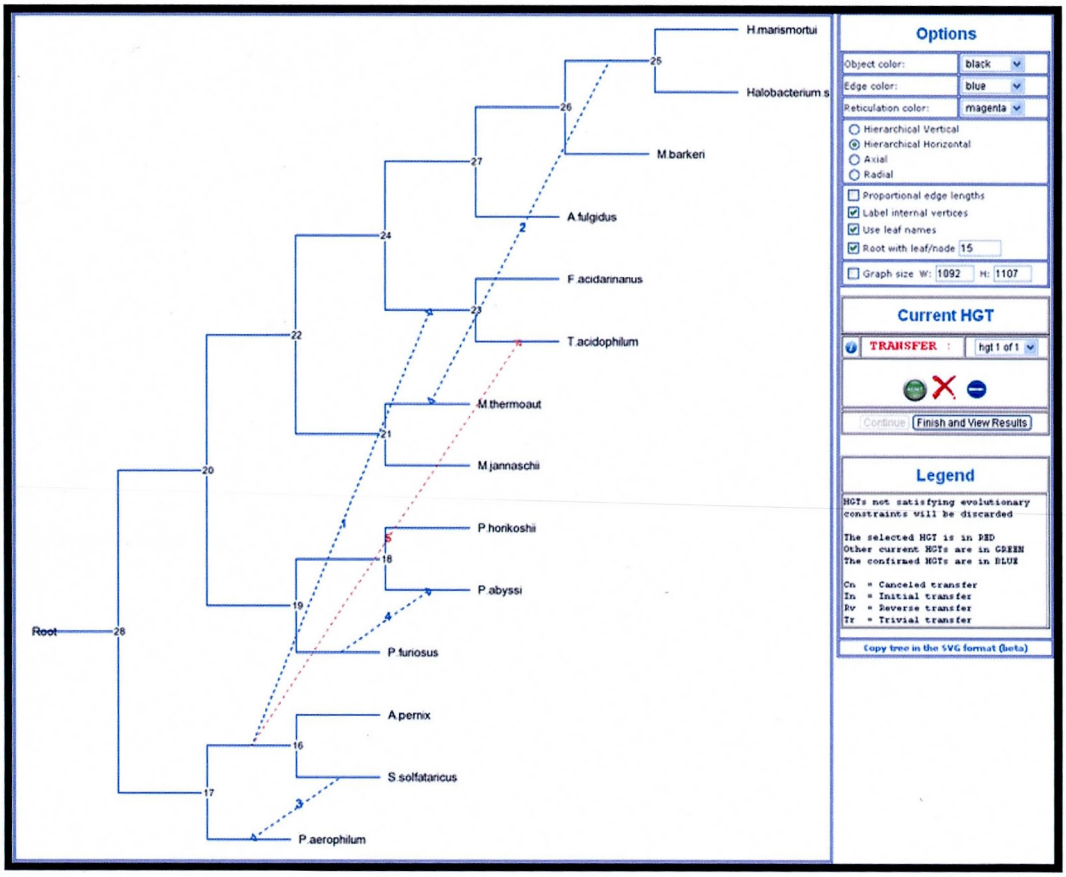


Figure 5.11 - Transfert (HGT_5) détecté à la seconde itération de la version interactive du programme de détection de THGs complets.

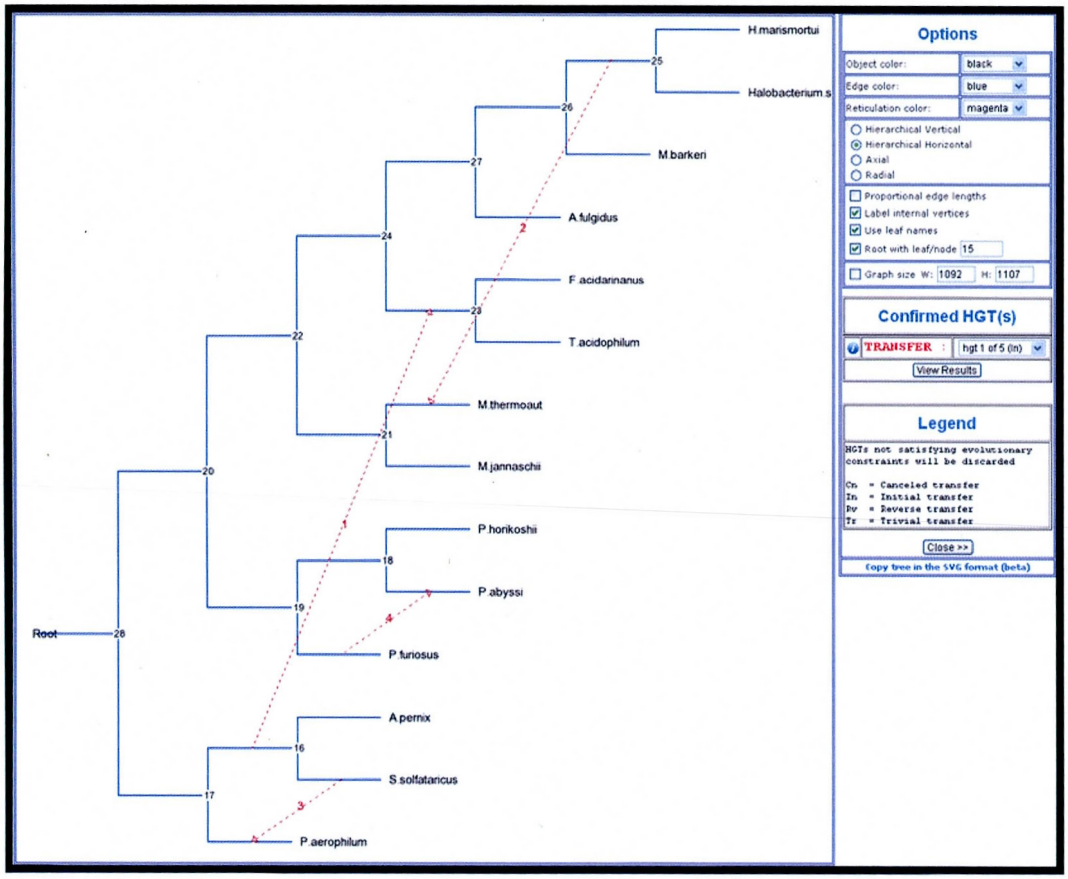


Figure 5.12 - Visualisation des résultats (scénario final validé) de la version interactive du programme de détection de THGs complets.

Les mêmes statistiques de détection, ainsi que les mêmes options de manipulation des résultats graphiques, que celles décrites dans la section 5.3, sont aussi disponibles à la suite de l'exécution de la version interactive du programme.

5.5 Interface de la *version consensus* du programme de détection de transferts horizontaux de gènes complets

La version consensus du programme de détection de THGs complet permet la détection de transferts à partir d'un arbre d'espèces unique et d'un ensemble d'arbres de gène, qui sont par exemple inférés à partir des séquences répliquées. Tous les THGs, apparaissant dans tous les

scénarios obtenus pour cet ensemble d'arbres de gène, sont identifiés et comptés. Pour plus d'informations, le lecteur est référé au chapitre II (section 2.6).

L'interface de la version consensus (voir la Figure 5.13) du programme de détection de THGs complets demande d'abord à l'utilisateur de spécifier l'arbre d'espèces ainsi que les arbres de gène codés en format Newick. Les options incluses dans l'interface sont les suivantes :

- la sélection du seuil de consensus (pourcentage minimal) à considérer;
- la sélection de la racine de l'arbre d'espèces (les racines des arbres de gène peuvent être spécifiées dans leur chaîne Newick; si elles ne sont pas spécifier, elles sont choisies par la stratégie *midpoint* présentée dans le chapitre II);
- le mode de détection de transferts horizontaux (un seul transfert détecté par itération ou plusieurs transferts détectés par itération);
- la sélection du critère d'optimisation : le critère des moindres carrés, la distance topologique de Robinson et Foulds ou la dissimilarité de bipartitions.

Évidemment, il est nécessaire de spécifier plusieurs arbres de gène pour pouvoir calculer un scénario de consensus des transferts. Dans l'exemple que nous présentons ci-dessous, nous avons répliqué l'arbre de gène dix fois, puis nous avons exécuté la version consensus (voir la Figure 5.13) du programme de détection de transferts horizontaux avec l'option du seuil de consensus de 40%.

En entrée nous avions donc :

- l'arbre d'espèces original de Matte-Tailliez *et al.* (2002), voir la Figure 5.2 :

  (((A.pernix:1.0,S.solfataricus:1.0):1.0,P.aerophilum:1.0):10.0,(((P.abyssi:1.0,P.horikoshii:1.0):1.0,P.furiosus:1.0):1.0,((M.jannaschii:1.0,M.thermoaut:1.0):1.0,((T.acidophilum:1.0,F.acidarinanus:1.0):1.0,(((Halobacterium.sp:1.0,H.marismortui:1.0):1.0,M.barkeri:1.0):1.0,A.fulgidus:1.0):1.0):1.0):1.0):1.0);

- et les dix arbres de gène répliqués :

(((((P.aerophilum:1.0,S.solfataricus:1.0):1.0,A.pernix:1.0):1.0,T.acidophilum:1.0):1.0,
F.acidarinanus:1.0):10.0,(((P.abyssi:1.0,P.furiosus:1.0):1.0,P.horikoshii:1.0):1.0,(((((H
alobacterium.sp:1.0,H.marismortui:1.0):1.0,M.thermoaut:1.0):1.0,M.barkeri:1.0):1.0,A
.fulgidus:1.0):1.0,M.jannaschii:1.0):1.0):1.0);

(((((P.aerophilum:1.0,S.solfataricus:1.0):1.0,A.pernix:1.0):1.0,T.acidophilum:1.0):1.0,
F.acidarinanus:1.0):10.0,(((P.abyssi:1.0,P.furiosus:1.0):1.0,P.horikoshii:1.0):1.0,(((((H
alobacterium.sp:1.0,M.barkeri:1.0):1.0,M.thermoaut:1.0):1.0,H.marismortui:1.0):1.0,A
.fulgidus:1.0):1.0,M.jannaschii:1.0):1.0):1.0);

(((((F.acidarinanus:1.0,S.solfataricus:1.0):1.0,A.pernix:1.0):1.0,T.acidophilum:1.0):1.0
,P.aerophilum:1.0):10.0,(((P.abyssi:1.0,P.furiosus:1.0):1.0,P.horikoshii:1.0):1.0,(((((Ha
lobacterium.sp:1.0,H.marismortui:1.0):1.0,M.thermoaut:1.0):1.0,M.barkeri:1.0):1.0,A.
fulgidus:1.0):1.0,M.jannaschii:1.0):1.0):1.0);

(((((P.aerophilum:1.0,S.solfataricus:1.0):1.0,A.pernix:1.0):1.0,M.jannaschii:1.0):1.0,F.
acidarinanus:1.0):10.0,(((P.abyssi:1.0,P.furiosus:1.0):1.0,P.horikoshii:1.0):1.0,(((((Hal
obacterium.sp:1.0,H.marismortui:1.0):1.0,M.thermoaut:1.0):1.0,M.barkeri:1.0):1.0,A.f
ulgidus:1.0):1.0,T.acidophilum:1.0):1.0):1.0);

(((((P.aerophilum:1.0,S.solfataricus:1.0):1.0,A.pernix:1.0):1.0,H.marismortui:1.0):1.0,
F.acidarinanus:1.0):10.0,(((P.abyssi:1.0,P.furiosus:1.0):1.0,P.horikoshii:1.0):1.0,(((((H
alobacterium.sp:1.0,T.acidophilum:1.0):1.0,M.thermoaut:1.0):1.0,M.barkeri:1.0):1.0,A
.fulgidus:1.0):1.0,M.jannaschii:1.0):1.0):1.0);

(((((P.abyssi:1.0,S.solfataricus:1.0):1.0,A.pernix:1.0):1.0,T.acidophilum:1.0):1.0,F.aci
darinanus:1.0):10.0,(((P.aerophilum:1.0,P.furiosus:1.0):1.0,P.horikoshii:1.0):1.0,(((((H
alobacterium.sp:1.0,H.marismortui:1.0):1.0,M.thermoaut:1.0):1.0,M.barkeri:1.0):1.0,A
.fulgidus:1.0):1.0,M.jannaschii:1.0):1.0):1.0);

(((((T.acidophilum:1.0,S.solfataricus:1.0):1.0,A.pernix:1.0):1.0,P.aerophilum:1.0):1.0,
F.acidarinanus:1.0):10.0,(((P.abyssi:1.0,P.furiosus:1.0):1.0,P.horikoshii:1.0):1.0,(((((H
alobacterium.sp:1.0,H.marismortui:1.0):1.0,M.thermoaut:1.0):1.0,M.barkeri:1.0):1.0,A
.fulgidus:1.0):1.0,M.jannaschii:1.0):1.0):1.0);

(((((P.aerophilum:1.0,S.solfataricus:1.0):1.0,A.pernix:1.0):1.0,T.acidophilum:1.0):1.0,F.acidarinanus:1.0):10.0,(((P.abyssi:1.0,P.furiosus:1.0):1.0,P.horikoshii:1.0):1.0,(((((Halobacterium.sp:1.0,A.fulgidus:1.0):1.0,M.thermoaut:1.0):1.0,M.barkeri:1.0):1.0,H.marismortui:1.0):1.0,M.jannaschii:1.0):1.0):1.0);

(((((P.aerophilum:1.0,M.jannaschii:1.0):1.0,A.pernix:1.0):1.0,T.acidophilum:1.0):1.0,F.acidarinanus:1.0):10.0,(((P.abyssi:1.0,P.furiosus:1.0):1.0,P.horikoshii:1.0):1.0,(((((Halobacterium.sp:1.0,A.fulgidus:1.0):1.0,M.thermoaut:1.0):1.0,M.barkeri:1.0):1.0,H.marismortui:1.0):1.0,S.solfataricus:1.0):1.0):1.0);

(((((P.aerophilum:1.0,S.solfataricus:1.0):1.0,P.abyssi:1.0):1.0,M.jannaschii:1.0):1.0,F.acidarinanus:1.0):10.0,(((A.pernix:1.0,P.furiosus:1.0):1.0,P.horikoshii:1.0):1.0,(((((Halobacterium.sp:1.0,H.marismortui:1.0):1.0,M.thermoaut:1.0):1.0,M.barkeri:1.0):1.0,A.fulgidus:1.0):1.0,T.acidophilum:1.0):1.0):1.0);

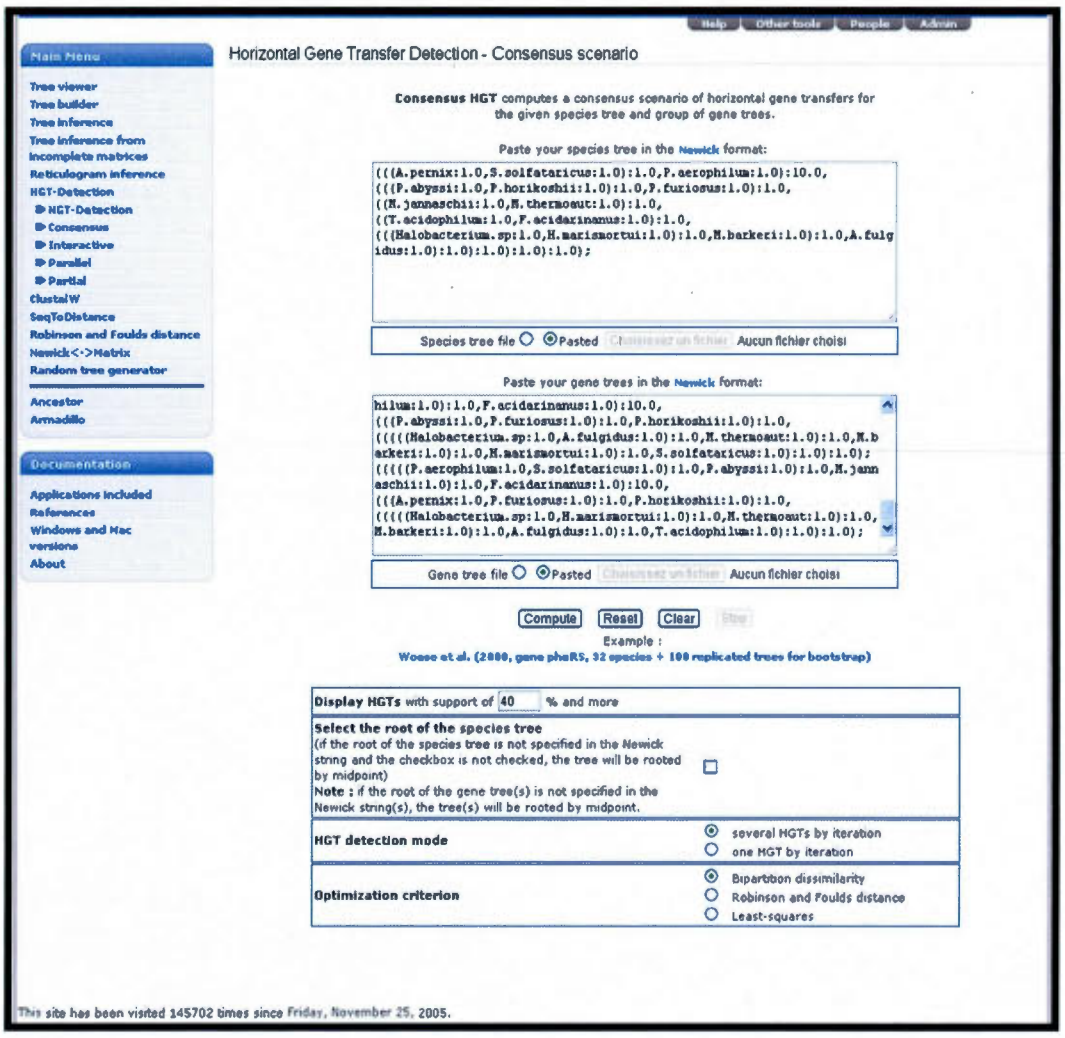


Figure 5.13 - Interface de la *version consensus* du programme de détection de THGs complets.

La Figure 5.14 montre les résultats de l'exécution de la version consensus du programme de détection de transferts pour le jeu de données considéré.

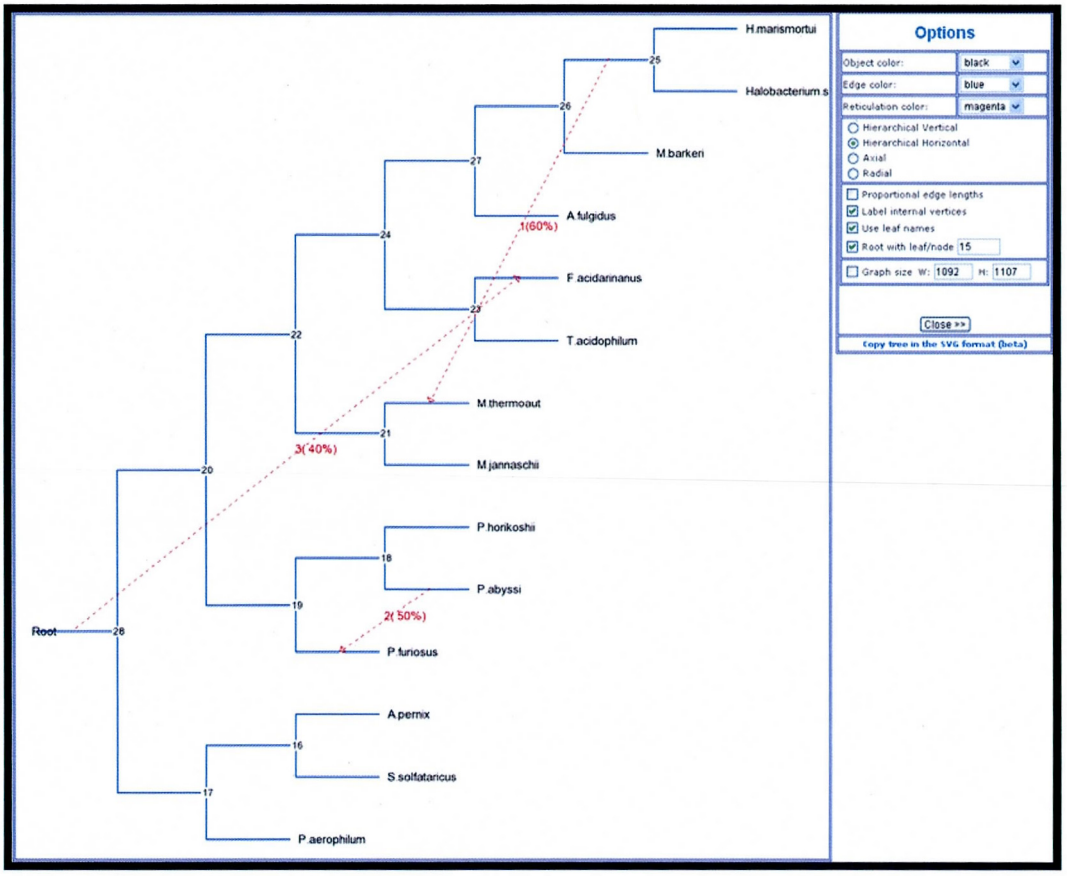


Figure 5.14 - Visualisation des résultats de la version consensus du programme de détection de THGs complets.

D'un autre coté, l'exécution du programme séquentiel de détection THGs complets avec l'option de bootstrap (voir la Figure 5.15) sur le même jeu de données, incluant l'arbre d'espèces de Matte-Tailliez *et al.* (2002) et les dix réplicats de l'arbre de gène, fournit les résultats exposés sur la Figure 5.16. On peut constater que les résultats obtenus par les versions *consensus* (voir la Figure 5.14) et *bootstrap* (voir la Figure 5.16) du programme de détection THGs sont différents (HGT_1 est le seul transfert commun retrouvé par les deux versions de l'algorithme). Notons qu'avec l'option de bootstrap de l'algorithme de détection de THGs complets, tous les calculs se font par rapport au premier arbre de gène spécifié, alors qu'avec la version consensus, tous les arbres de gène sont considérés comme équivalents pour le calcul du score de consensus des THGs.

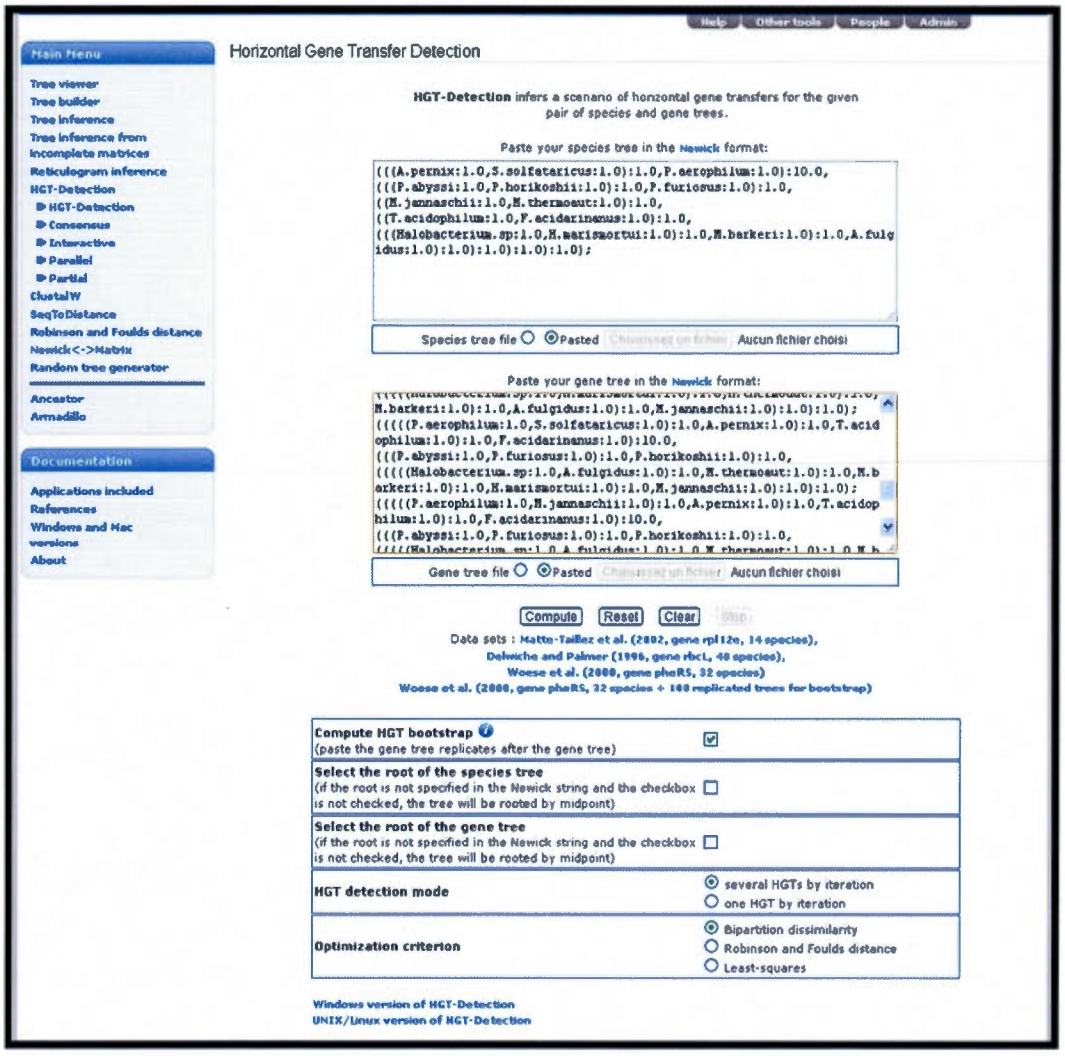


Figure 5.15 - Interface Web du programme séquentiel de détection de THGs complets avec l'option de bootstrap.

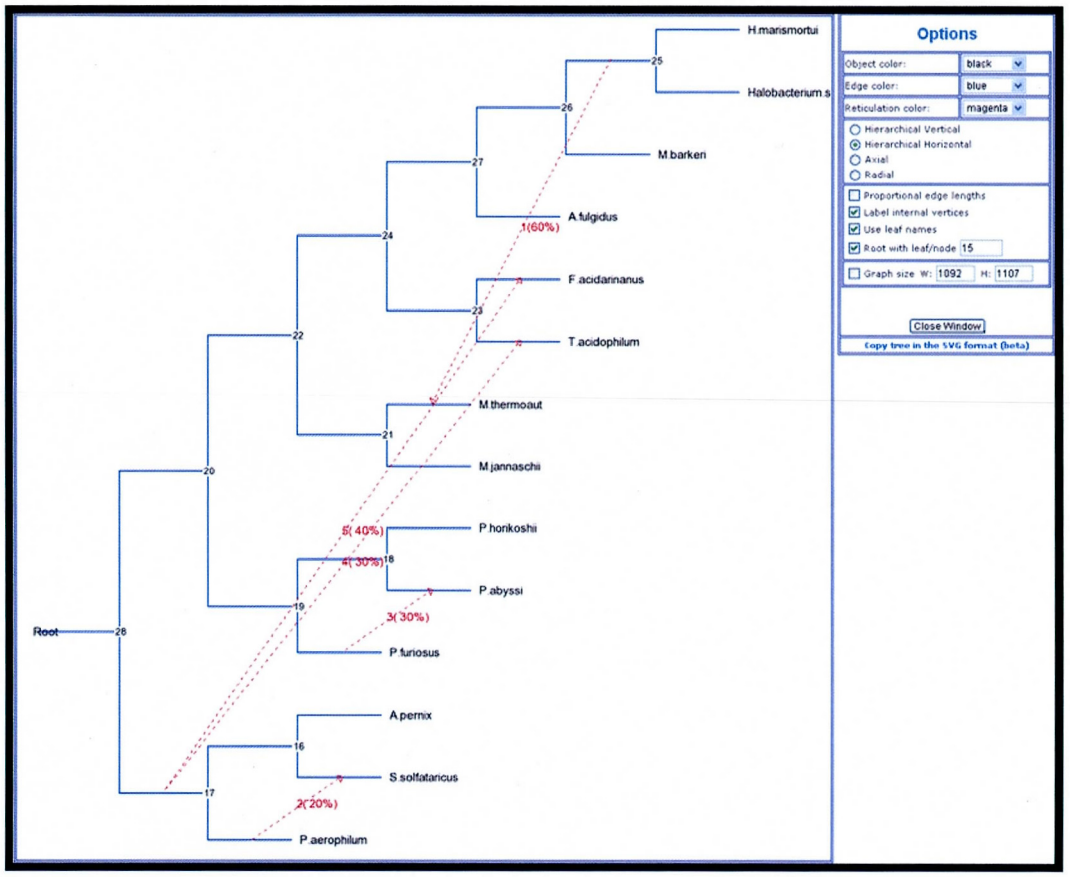


Figure 5.16 - Visualisation des résultats du programme séquentiel de détection de THGs complets avec l'option de bootstrap.

5.6 Interface du programme de détection de *transferts horizontaux de gènes partiels*

Le programme séquentiel de détection de THGs partiels procède par une réconciliation progressive d'une phylogénie enracinée d'espèces avec plusieurs phylogénies enracinées de gène qui peuvent être inférées à partir de l'alignement des séquences multiples donné (voir le chapitre III pour plus de détails sur le deuxième algorithme de détection de THGs partiels dont l'interface Web est présentée dans cette section). Cette interface (voir la Figure 5.17)

permet à l'utilisateur de spécifier l'arbre d'espèces (codé en format Newick) et les séquences de gène pour les mêmes espèces considérées. L'interface inclut les options suivantes :

- une adresse e-mail pour aviser l'utilisateur de la fin du traitement; le deuxième algorithme de détection de THGs partiels est beaucoup plus gourmand en terme de temps d'exécution que les algorithmes de détection de THGs complets;
- le mode d'inférence d'arbre (en utilisant les méthodes NJ ou PhyML);
- le type des séquences à traiter (l'ADN ou les acides aminés);
- la taille de la fenêtre coulissante;
- la taille du pas de progression;
- le nombre de réplicats dans le bootstrap;
- le seuil de bootstrap (pourcentage minimal) à considérer;
- le nombre de fenêtres consécutives nécessaires pour valider un transfert.

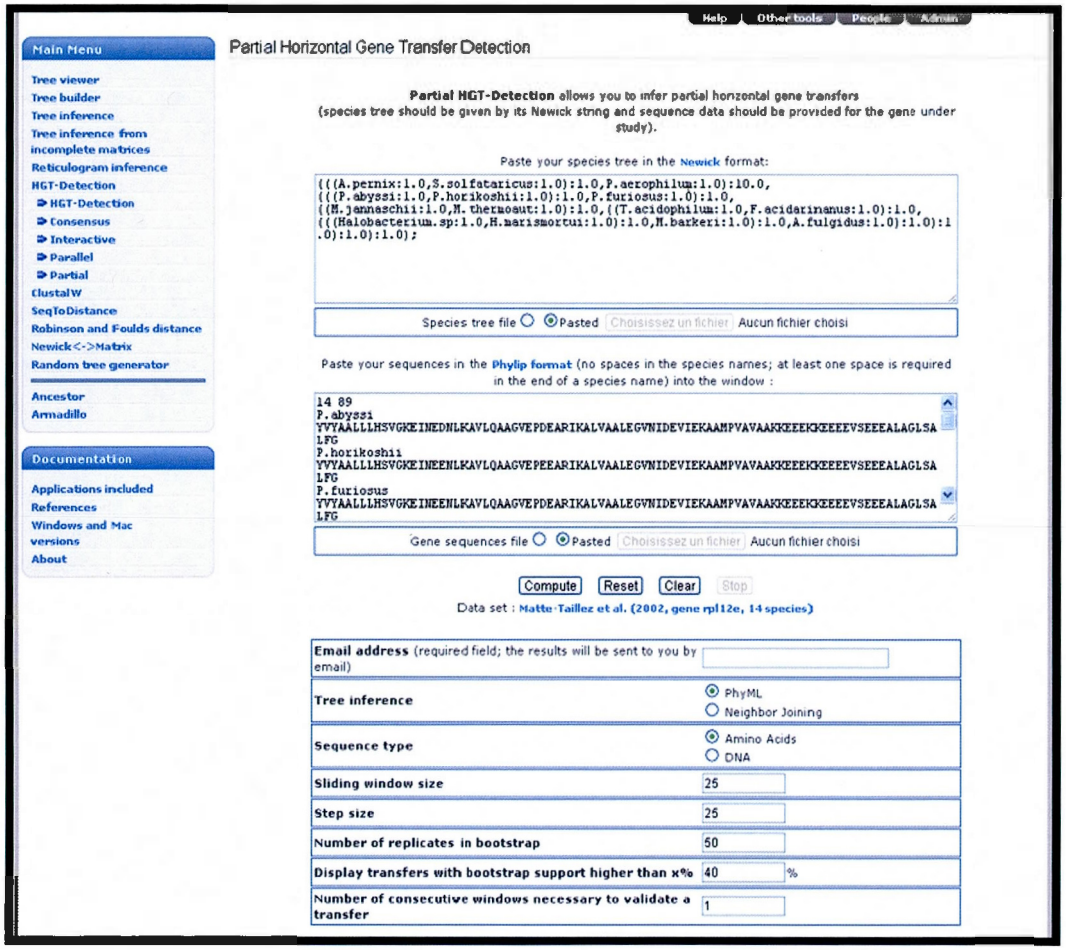


Figure 5.17 - Interface Web du programme de détection de THGs partiels.

Les transferts partiels détectés pour le jeu de données de Matte-Tailliez *et al.* (2002) sont présentés dans la Figure 5.18. Quatre des cinq transferts décrits dans la section 5.1 ont été retrouvés par le programme de détection de THGs partiels sur les différents intervalles de l'alignement de séquences donné. En plus, deux nouveaux transferts horizontaux partiels qui ne figurent pas parmi les THGs complets ont été retrouvés (voir la Figure 5.18).

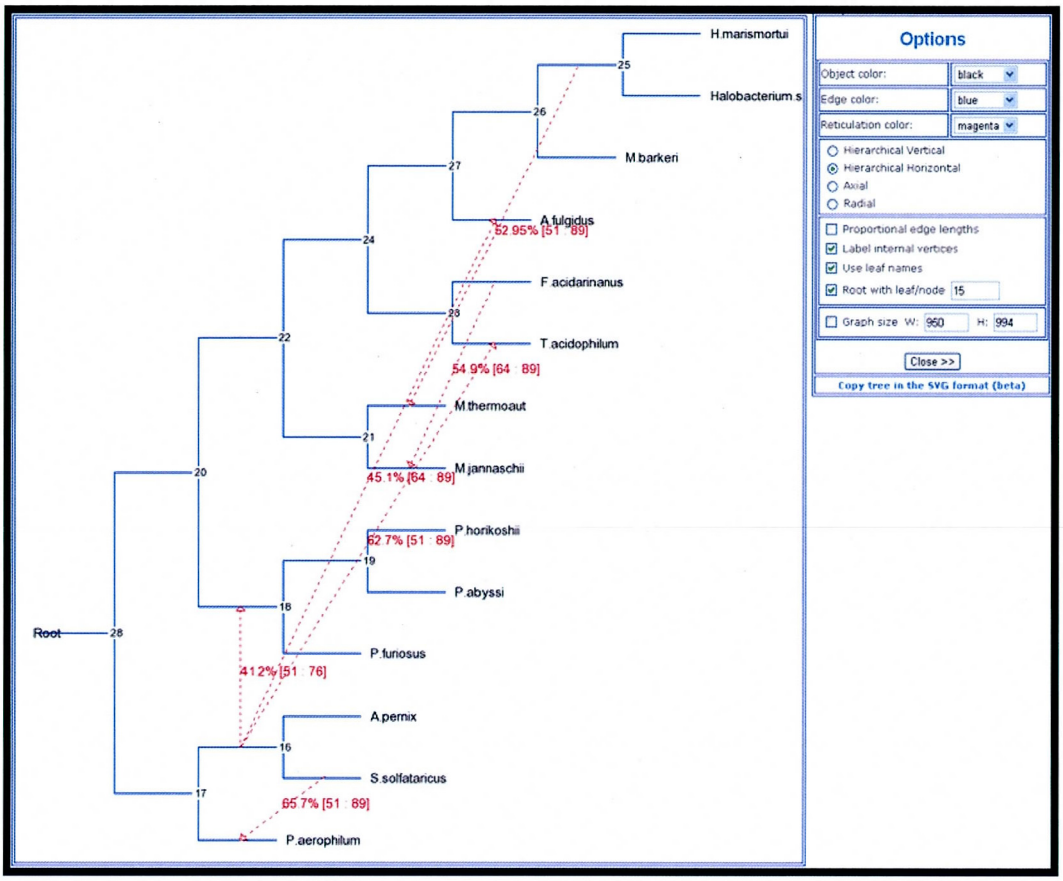


Figure 5.18 - Visualisation des résultats du programme de détection de transferts horizontaux partiels. Les pourcentages de bootstrap des THGs obtenus sont suivis des intervalles de l'alignement de séquences données, affectés par ces transferts (présentés entre les crochets).

Les mêmes statistiques de détection, ainsi que les mêmes options de manipulation des résultats graphiques, que celles qui sont disponibles pour la détection de transferts complets, sont aussi disponibles à la suite de l'exécution du programme de détection de THGs partiels.

5.7 Résumé

Dans ce chapitre, nous avons présenté les interfaces des différentes versions des programmes de détection de transferts horizontaux de gènes complets et partiels. Ces interfaces ont été développées et ajoutées à la plateforme d'analyse phylogénétique *T-Rex* (Makarenkov, 2001). Pour tester et présenter ces différentes interfaces, nous avons utilisé un jeu de données réelles

concernant l'évolution du gène *rpl12e* (Matte-Tailliez *et al.*, 2002). Ce jeu de données nous a permis d'exhiber différentes solutions possibles, obtenues en fonction des connaissances biologiques *a priori* sur les données à traiter.

Notons que le site Web de *T-Rex* disponible à l'URL www.trex.uqam.ca devient de plus en plus populaire auprès des bioinformaticiens du monde entier. À ce jour (février 2012), notre site compte plus de 140,000 visiteurs depuis novembre 2005. Nous avons écrit un article (Boc, Diallo et Makarenkov, 2012) décrivant les propriétés de notre site Web. Cet article a été accepté pour publication dans la revue *Nucleic Acids Research* (Web Server Issue).

CONCLUSIONS ET PERSPECTIVES

Dans cette thèse, nous avons présenté de nouveaux algorithmes, programmes et interfaces Web servant à la détection de transferts horizontaux de gènes *complets* et *partiels*. Nos contributions principales lors de la réalisation de ce projet doctoral sont les suivantes :

Première contribution

D'abord, nous avons mis au point un algorithme de détection de transferts horizontaux de gènes complets basé sur une réconciliation graduelle d'une phylogénie (*i.e.*, arbre phylogénétique représentant l'évolution d'un groupe d'espèces donné) enracinée d'espèces et d'une phylogénie enracinée d'un gène. Nous y avons inclus les procédures de la prise en compte de certaines règles d'évolution et de la validation par bootstrap des transferts détectés. Nous avons aussi défini la dissimilarité de bipartitions et avons établi certaines de ses propriétés. Différentes versions de cet algorithme ont été développées : version standard, version interactive et version consensus. La version interactive a été mise en place afin d'intégrer un processus de validation des transferts détectés et d'augmenter la plausibilité biologique des résultats de l'algorithme; la détection de transferts horizontaux de gènes dans la version standard se fait automatiquement, sans connaissances a priori sur les données biologiques traitées. La version consensus, quant à elle, permet de retrouver un scénario de consensus de tous les transferts détectés pour un ensemble d'arbres de gène donné. Les simulations Monte Carlo servant à valider les résultats de la nouvelle méthode de détection de THGs complets ont été effectuées.

Deuxième contribution

Nous avons ensuite présenté une généralisation du modèle de THGs complets applicable à un modèle en réseau où l'on considère le transfert partiel d'un gène. Ce modèle, qui suppose la formation de gènes mosaïques, est beaucoup plus complexe que le modèle du transfert complet car la distance minimale entre deux espèces peut être calculée à travers plusieurs

chemins possibles. Deux approches ont été proposées. La première approche se base sur l'optimisation des distances par les moindres carrés afin de retrouver les transferts horizontaux optimaux ainsi que les portions transférées du gène. Nous avons démontré que ce problème d'optimisation est NP-difficile (dans le cas de l'optimisation par les moindres carrés). La deuxième approche se base sur une procédure de fenêtre coulissante qui analyse des fragments d'un alignement de séquences. Pour chaque position fixe de cette fenêtre, un arbre de gène partiel est inféré et un scénario de THGs est calculé en réconciliant l'arbre de gène partiel et l'arbre d'espèces donné. Les simulations Monte Carlo servant à valider les résultats des deux nouveaux algorithmes de détection de THGs partiels, incluant leur comparaison, ont été réalisées.

Troisième contribution

Nous avons développé une version parallèle du programme de détection de transferts horizontaux de gènes complets, *HGT-Detection* (voir l'Annexe C.3). À travers les simulations effectuées, nous avons montré les gains de performances obtenus par la version parallèle du programme. La comparaison avec la version séquentielle du programme a été effectuée sur des jeux de données de tailles variables incluant 10 et 20 transferts.

Nous avons aussi mis en place une plateforme d'analyse phylogénétique : la version Web du logiciel *T-Rex*. La version Web du logiciel *T-Rex* inclut plusieurs applications bioinformatiques servant à l'analyse, la reconstruction et la visualisation des phylogénies. Des applications développées par des chercheurs de renom, telles que certains algorithmes du paquet *PHYLIP* (Felsenstein, 1989) ou encore le programme d'alignement de séquences *ClustalW* (Thompson, 1994), ont été d'abord ajoutées à ce site Web. Nous avons ensuite intégré à cette plateforme les interfaces des programmes de détection de transferts horizontaux de gènes, qui n'étaient pas incluses dans la version *Windows* du logiciel *T-Rex* (Makarenkov, 2001). Les langages *HTML* et *PHP* ont été utilisés pour le développement du site Web et la gestion des données provenant des formulaires. Des scripts écrits en langage *Perl* ont été utilisés pour récupérer les paramètres d'exécution du site Web et pour encapsuler l'exécution des programmes externes écrits en langages *C/C++*. L'adresse URL du site Web de *T-Rex* est la suivante : http://www.trex.uqam.ca. Ce site se trouve en perpétuelle

évolution : depuis sa mise en place officielle en novembre 2005, il a reçu plus de 140,000 visites (données de février 2012).

Finalement, une nouvelle méthode et une interface Web correspondante ont été développées pour faciliter la tâche du programmeur lors de la transformation d'un programme séquentiel en un programme parallèle. Nous avons identifié les principales étapes de cette transformation en utilisant la bibliothèque d'échange de messages MPI comme structure de soutien. À chaque étape de cette transformation, nous avons proposé une solution afin de simplifier le travail du programmeur, diminuer le temps nécessaire au développement du code parallèle et d'augmenter sa qualité du programme parallèle. Nous avons mis en place une interface Web (disponible à l'adresse URL suivante : http://trex_cluster.labunix.uqam.ca) afin de permettre l'utilisation pratique des outils de parallélisation développés.

Perspectives

Parallélisation du deuxième algorithme de détection de transferts horizontaux de gènes partiels

Le deuxième algorithme de détection de THGs partiels a été développé à travers le langage de script *Perl*, qui invoque les fonctions principales suivantes : *Robinson and Foulds* - pour le calcul de la distance topologique de *Robinson et Foulds*, PhyML - pour l'inférence d'arbres et *HGT-Detection* - pour l'inférence de scénarios des THGs associés aux différents intervalles de l'alignement de séquences considérés (voir la Figure 3.6 pour plus d'informations sur le schéma algorithmique complet du deuxième algorithme de détection de THGs partiels). Une parallélisation du deuxième algorithme de détection de THGs partiels (voir la Figure 3.6) pourrait se faire des deux manières suivantes :

- L'exécution en parallèle de la boucle *Pour* : chaque position fixe de la fenêtre coulissante considérée serait traitée indépendamment des autres, où le nombre de processus indépendants pourrait être déterminé par l'expression suivante :

  $(l - w)/s$;

- La parallélisation des algorithmes PhyML (voir Guindon *et al.*, 2010 concernant la parallélisation de cet algorithme), *HGT-Detection* (voir le chapitre IV, section 4.7, concernant la parallélisation de cet algorithme) et *Robinson et Foulds* (Robinson et Foulds, 1981) pourrait aussi accélérer l'exécution de la version parallèle du programme de façon significative.

Notons que la première manière de la parallélisation mentionnée apporterait un gain plus important grâce à une plus grande portion parallélisable de l'algorithme. Une telle parallélisation permettrait, entre autres, d'implanter une procédure utilisant une fenêtre coulissante de longueur variable (cette procédure aurait une complexité algorithmique supérieure à celle du deuxième algorithme de détection de THGs que nous avons présenté dans cette thèse), qui s'adapterait encore mieux aux tailles variables des transferts partiels.

Il serait aussi intéressant de se pencher sur la parallélisation de la version consensus et de celle utilisant le bootstrap.

Amélioration de la méthode de transformation semi-automatique d'un programme séquentiel en un programme parallèle

Nous pourrions également améliorer notre méthode et notre interface Web, servant à la transformation de programmes séquentiels en programmes parallèles, des manières suivantes :

- Développer et ajouter de nouveaux procédés pour la gestion des analyses des dépendances des données, ainsi que pour la traduction des structures de données *C* en structures de données MPI;
- Inclure d'autres environnements de programmation parallèle, tels que OpenMP;
- Paralléliser d'autres structures de contrôle, telles que la boucle *while*;
- Ajouter la possibilité d'une transformation semi-automatique à partir d'autres langages de programmation, tels que *Java*;
- Inclure des outils pour effectuer automatiquement les tests de non régression du programme parallèle développé;

- Inclure des outils pour effectuer le profilage des programmes séquentiels à paralléliser.

Programmation GPU

Le GPU, qui est un microprocesseur présent sur les cartes graphiques d'un ordinateur ou d'une console de jeux vidéo, peut exécuter (à travers un processus de délégation) une partie du travail habituellement effectuée par le processeur principal (CPU). Le GPU n'était initialement dédié qu'aux opérations d'affichage et de manipulation de données graphiques. Grâce à une structure hautement parallèle, les processeurs des cartes graphiques modernes sont très performants en termes de puissance de calcul.

Au début de notre projet doctoral, nous avons effectué des tests impliquant les GPU de *Nvidia GeForce 9800 GX2* sur des programmes parallèles bioinformatiques. Ces tests, qui ne sont pas présentés dans la thèse, n'ont pas donné de résultats concluants du fait que la mémoire locale du GPU n'était pas assez volumineuse pour le traitement efficace des programmes bioinformatiques considérés. Cependant, il faut noter que l'API CUDA (Compute Unified Device Architecture) de Nvidia était, à ce moment, à sa première version et contenait de nombreux bogues. Avec les nouvelles cartes GPU (telles que la carte *Nvidia GeForce GTX 580* ou la carte *ATI FirePro V9800*) plus rapides et disposant d'une plus grosse mémoire, ainsi que la conception de l'API OpenCL (Open Computing Language), nous pourrions nous pencher à nouveau sur ce projet. Notons que la méthode de parallélisation utilisant les GPU, basée sur l'architecture SIMD, sera différente de la méthode de parallélisation MPI considérée dans cette thèse, qui est basée sur l'architecture MIMD.

ANNEXE A

ARTICLES PUBLIÉS ET SOUMIS POUR PUBLICATION

L'annexe A présente les articles publiés et soumis pour publication qui ont été rédigés dans le cadre de ce projet doctoral. Ma contribution à chaque publication est spécifiée.

Makarenkov, V., Boc, A. et **Diallo, Alpha B.** (2007), La dissimilarité de bipartitions et son utilisation pour détecter les transferts horizontaux de gènes, *Actes des 14èmes Rencontres de la Société Francophone de Classification, ENST de Paris, France*, pages 90-93.

Contribution de l'étudiant :

- Développement de l'algorithme de détection des transferts horizontaux complets, définition et description de la dissimilarité de bipartitions (avec A. Boc).
- Réalisation de tests de validation pour la dissimilarité de bipartitions.
- Simulations effectuées pour ce modèle.
- Cet article a été rédigé de façon conjointe.

Makarenkov, V., Boc, A. et **Diallo, Alpha B.** et Diallo, Abdoulaye B. (2008), Algorithms for detecting horizontal gene transfers: Theory and practice, In *Data Mining and Mathematical Programming*, P.M. Pardalos et P. Hansen (Eds.), CRM Proceedings and AMS Lecture Notes, volume 45, pages 159-179.

Contribution de l'étudiant :

- Participation à la définition et à la preuve de NP-complétude du problème de recherche des transferts horizontaux partiels en utilisant le modèle des moindres carrés.
- Développement du modèle des transferts horizontaux partiels basé sur l'optimisation par des moindres carrés (avec A. Boc).
- Simulations effectuées pour ce modèle.
- Cet article a été rédigé de façon conjointe.

Boc, A., **Diallo, Alpha B.** et Makarenkov, V. (2011), Un nouvel algorithme pour la détection de transferts horizontaux de gènes partiels entre les espèces et pour la classification des transferts inférés, *Actes des 18èmes Rencontres de la Société Francophone de Classification*, Orléans, France, pages 25-28.

Contribution de l'étudiant :

- Développement de l'approche utilisant la fenêtre coulissante pour la détection de transferts de gènes partiels.
- Simulations et application de la nouvelle méthode aux différents jeux de données.
- Participation au développement du modèle de validation des transferts partiels par bootstrap.
- Cet article a été rédigé de façon conjointe.

Diallo, Alpha B., Lord, E. et Makarenkov, V. (2012), A new web server for parallelization of bioinformatics algorithms. Soumis (12 pages).

Contribution de l'étudiant :

- Développement et installation des différents outils d'automatisation et de parallélisation.
- Développement de l'interface de parallélisation semi-automatique.
- Cet article a été rédigé de façon conjointe.

Boc, A., **Diallo, Alpha B.** et Makarenkov, V. (2012), T-Rex : a web server for inferring, validating and visualizing phylogenetic trees and networks. À paraître dans *Nucleic Acids Research* (Web Server Issue).

Contribution de l'étudiant :

- Développement des différentes versions du programme de détection des transferts horizontaux de gènes complets et partiels, dont les versions parallèles.
- Réalisation des tests de fiabilité et de performance.
- Mise en place des interfaces Web des différentes applications bioinformatiques.
- Cet article a été rédigé de façon conjointe.

Dans la plupart de ces publications, l'ordre des auteurs correspond à l'ordre alphabétique de leur nom de famille, à l'exception des articles des conférences Makarenkov *et al.* (2007) et Makarenkov *et al.* (2008) où mon directeur de recherche, M. Vladimir Makarenkov, est allé présenter les résultats de nos travaux.

ANNEXE B

OUTILS DE TRANSFORMATION DES STRUCTURES DE DONNÉES, OUTILS DE DÉTECTION DES DÉPENDANCES DES DONNÉES, GÉNÉRATEUR DE COMPILATEURS SABLECC ET MÉTHODE DE DIVISION DES TÂCHES

L'annexe B débute par la présentation des outils utilisés pour la transformation des structures de données *C* en structures de données MPI. Ensuite, nous présentons les outils utilisés pour l'analyse des dépendances des données et le générateur de compilateurs *SableCC*. Nous terminons cette annexe par la présentation d'une méthode supplémentaire pour la division des tâches à exécuter en parallèle.

B.1 Outils utilisés pour la transformation des structures de données C en structures de données MPI

Nous avons automatisé la transformation des structures de données *C* en structures de données MPI à travers l'utilisation des deux outils suivants : *Automap* (Devaney *et al.*, 1997) et *Autoserial* (Schaeli *et al.*, 2008).

B.1.1 Automap

Automap (Devaney *et al.*, 1997) est un outil qui facilite la création de structures de données MPI. C'est un compilateur qui traduit automatiquement des structures de données *C* en structures de données MPI. Cet outil requiert en paramètre d'entrée un fichier qui définit les

structures de données du programme séquentiel à transformer (voir le Tableau B.1 pour un exemple pratique), puis il génère deux fichiers principaux (*mpitypes.h* et *mpitypes.inc*) et des fichiers *logs* contenant des informations sur le déroulement du processus de traduction des structures de données. Le premier fichier généré par *Automap*, *mpitypes.h*, définit des fonctions et des prototypes MPI. Le second fichier, *mpitypes.inc*, contient les structures de données MPI ainsi que les fonctions requises pour effectuer l'allocation et la désallocation de ces structures de données.

Le fichier d'entrée qui contient les structures de données à transformer doit inclure plusieurs mots clés qui guideront la traduction. Ces mots clés sont les suivants : le mot clé /*~ *AM_Begin* */ pour définir le début du fichier à transformer, le mot clé /*~ *AM_End* */ pour définir la fin du fichier à transformer et le mot clé /*~ AM */ pour identifier les structures de données à transformer. Ces différents mots clés font parties intégrantes de la grammaire de ce compilateur. Les structures de données transformées à travers *Automap* peuvent contenir des tableaux, des pointeurs et peuvent même inclure d'autres structures de données. Cependant, *Automap* ne permet pas une définition de structures de données avec l'instruction *struct*. La définition de structures de données doit se faire obligatoirement avec l'instruction *typedef struct*.

Le Tableau B.1 présente un exemple d'un fichier d'entrée de *Automap* :

```
/*~ AM_Begin */
typedef struct {
int a;
double d[6];
char b[7];
int c[8];
} Partstruct/*~ AM */;

typedef struct {
int a;
double d[6];
char b[7][9][10];
int c[8];
} Partstruct2/*~ AM */;

/*~ AM_End */
```

Tableau B.1 - Exemple d'un fichier d'entrée de Automap. Les mots clés importants sont mis en gras.

Les options de la commande *Automap* sont les suivantes :

```
AutoMap [-help] [-v] [-log] [-noAL] filename
        -help      : Will print this help menu
        -v         : Verbose mode
        -log       : Will generate the "logbook.txt" for this run
        -noAL      : Will not generate the entries for use with AutoLink
        Filename   : name of the C typedef definition file to analyze
```

Tableau B.2 - Les options de la ligne de commande de *Automap*.

B.1.2 Autoserial

Autoserial (Schaeli *et al.*, 2008) est une bibliothèque *C++* permettant la déclaration d'objets sérialisables qui peuvent être facilement sauvegardés dans des fichiers ou envoyés sur le réseau. L'avantage majeur de cet outil est que le code de la sérialisation et de la désérialisation est généré automatiquement, ce qui rend la bibliothèque très facile à utiliser. *Autoserial* est capable de sérialiser de nombreux types de données différents comme des pointeurs simples, des tableaux statiques et dynamiques, etc. Cette bibliothèque permet, à travers une technique d'emballage (*wrapper*), l'envoie et la réception des objets sérialisables à l'aide des fonctions de communication point à point *AS_MPI_Send* et *AS_MPI_Recv* (voir leurs prototypes dans le Tableau B.3).

```
Prototypes
// MPI_Send wrapper
int AS_MPI_Send(autoserial::ISerializable &obj, int dest, int tag, MPI_Comm comm)

// MPI_Recv wrapper
// The wrapper creates and initializes the object internally. It is therefore the
// responsibility of the caller to delete the created object
int AS_MPI_Recv(autoserial::ISerializable *&obj, int src, int tag, MPI_Comm comm, MPI_Status *st);
```

Tableau B.3 - Les prototypes des fonctions *AS_MPI_Send* et *AS_MPI_Recv* de la bibliothèque *Autoserial*.

Pour utiliser cette bibliothèque, il est nécessaire d'inclure le fichier *autoserial_mpi.h*. Le Tableau B.4 présente un exemple de fichier utilisant la bibliothèque *Autoserial*.

```
#include <autoserial/autoserial_mpi.h>


...


// Create an object on process 0, and send it to process 1
MySerializableObject *obj;
if (rank == 0)
{
    obj = new MySerializableObject();           // Create object
    AS_MPI_Send(*obj, 1, 0, MPI_COMM_WORLD);    // Send object from 0 to 1
}
else
{
    AS_MPI_Recv(obj, 0, 0, MPI_COMM_WORLD);     // Receive object from 0
}
delete obj;                                     // Both processes must delete obj
```

Tableau B.4 - Exemple d'un fichier utilisant la bibliothèque *Autoserial*.

B.2 Outils utilisés pour la détection des dépendances des données

Nous avons procédé à l'analyse des dépendances des données à travers la bibliothèque PLATO (Psarris et Kyriakopoulos, 2004; Kyriakopoulos, 2007) et le compilateur *Cetus* (Johnson *et al.*, 2005; Dave *et al.*, 2009).

B.2.1 PLATO

Les techniques d'analyse des dépendances développées dans le cadre du projet de la librairie PLATO peuvent traiter des cas complexes de détection des dépendances des données. Ces techniques sont basées sur un ensemble de méthodes qui peuvent, en temps polynomial, confirmer ou réfuter les dépendances dans le code source d'un programme donné incluant : des expressions non-linéaires ou symboliques, des bornes de boucles complexes, des tableaux ayant des indices couplés, ainsi que l'instruction conditionnelle *if*.

Les trois types de tests possibles avec PLATO sont les suivants :

- Le premier test, LVI (Linear Variable Interval), est applicable quand l'équation de résolution des dépendances des données est linéaire.
- Le second test, PVI (Polynomial Variable Interval), est applicable quand l'équation de résolution des dépendances des données est polynomiale.
- Le dernier test, RVI (Rational Variable Interval), est applicable quand l'équation de résolution des dépendances des données est rationnelle.

Pour effectuer l'analyse des dépendances des données avec PLATO, il faut d'abord écrire un fichier *main* en *C* qui spécifie le test des dépendances à effectuer. Par exemple pour une boucle *for*, nous devons spécifier entre autres l'indice de début, l'indice de fin, le pas de la boucle, etc. La compilation de ce fichier est ensuite effectuée et son exécution permet d'obtenir les résultats de l'analyse des dépendances. La Tableau B.6 met en évidence un exemple d'un fichier *main* faisant appels aux fonctions de la librairie PLATO pour l'analyse des dépendances des données de la boucle présentée dans le Tableau B.5 (dans le cadre d'un test PVI). La compilation de ce fichier *main* avec d'autres classes de la librairie est ensuite

effectuée. L'exécution de ce programme permet de réaliser l'analyse des dépendances des données.

```
void main()
{
        // Déclaration des variables
        int i,n=100;
        int A[100], B[100];

        for (i=1;i<n;i++){
                A[i] = A[i-1]+B[i];
        }
}
```

Tableau B.5 - Exemple d'un fichier *C* contenant une boucle *for*.

```
#include "PolynomialVITestProblem.h"
#include "DDTesting.h"
#include "MathUtils.h"

using namespace plato;

class DDPolynomialVITestProblem : public PolynomialVITestProblem {
    public:
        DDPolynomialVITestProblem();
        virtual ~DDPolynomialVITestProblem();
};

DDPolynomialVITestProblem::DDPolynomialVITestProblem() : PolynomialVITestProblem (true) {}

DDPolynomialVITestProblem::~DDPolynomialVITestProblem() {}

// Définition de la boucle à tester
class ComparePolynomialVITestProblem : public DDPolynomialVITestProblem {
    private:
Variable i,n;
    public:
        CompareLinearVITestProblem();
        virtual int getLoopNestSize() const;
};

// Spécification des paramètres de la boucle
ComparePolynomialVITestProblem::ComparePolynomialVITestProblem(): i("i"),n("n"){
    addSymbolicVariable(n);
    addCommonLoopVariable(i,1,n,+1);
    addSubscript(i-1 , i);
}

// Niveau d'imbrication de la boucle
int ComparePolynomialVITestProblem::getLoopNestSize() const {
    return 1;

}

// Exécution du test des dépendances des données
#include "PolynomialExpression.h"

int main()
{
   Variable::resetVars();
   ComparePolynomialVITestProblem problem;
   DVHierarchyDriver dr(problem, true);
   dr.doDDTest();
   return 0;
}
```

Tableau B.6 - Fichier *main* pour l'exécution de l'analyse des dépendances des données de la boucle en utilisant la librairie PLATO, présentée au Tableau B.5. Les instructions les plus importantes sont mises en gras.

B.2.2 Cetus

Le compilateur *Cetus* (Johnson *et al.*, 2005; Dave *et al.*, 2009) est une infrastructure *Java* qui permet une parallélisation automatique de programme séquentiels à travers OpenMP. Cette infrastructure inclut aussi deux méthodes d'analyse des dépendances des données : *Banerjee-Wolfe* (Wolfe et Banerjee, 1987) et *Range test* (Blume et Eigenmann, 1994).

Nous pouvons exécuter cet outil à travers la commande suivante :

```
java -classpath ".:cetus.jar:antlr.jar" cetus.exec.Driver -normalize-loops test.c
```

L'écriture d'un fichier *main* en *C* qui spécifie l'analyse des dépendances n'est pas nécessaire, ce qui représente un avantage majeur par rapport à PLATO. Les différentes options de *Cetus* sont présentées dans le Tableau B.7 ci-dessous.

```
cetus.exec.Driver [option]... [file]...
--------------------------------------------------------------------------------
UTILITY
--------------------------------------------------------------------------------
-debug_parser_input
    Print a single preprocessed input file before sending to parser and exit

-debug_preprocessor_input
    Print a single pre-annotated input file before sending to preprocessor and exit

-dump-options
    Create file options.cetus with default options

-dump-system-options
    Create system wide file options.cetus with default options

-expand-all-header
    Expand all header file #includes into code

-expand-user-header
    Expand user (non-standard) header file #includes into code

-help
    Print this message

-load-options
    Load options from file options.cetus

-macro
    Sets macros for the specified names with comma-separated list (no space is allowed).
    e.g., -macro=ARCH=i686,OS=linux

-outdir=dirname
    Set the output directory name (default is cetus_output)

-parser=parsername
    Name of parser to be used for parsing source file

-preprocessor=command
    Set the preprocessor command to use


-preserve-KR-function
    Preserves K&R-style function declaration
```

```
-skip-procedures=proc1,proc2,...
    Causes all passes that observe this flag to skip the listed procedures

-verbosity=N
    Degree of status messages (0-4) that you wish to see (default is 0)

-version
    Print the version information

--------------------------------------------------------------------------------
ANALYSIS
--------------------------------------------------------------------------------
-alias=N
    Specify level of alias analysis
      =0 disable alias analysis (assume no alias)
      =1 advanced interprocedural analysis (default)
         Uses interprocedural points-to analysis
      =2 assume no alias when points-to analysis is too conservative

-callgraph
    Print the static call graph to stdout

-ddt=N
    Perform Data Dependence Testing
      =1 banerjee-wolfe test (default)
      =2 range test

-parallelize-loops
    Annotate loops with Parallelization decisions
      =1 parallelizes outermost loops (default)
      =2 parallelizes every loop
      =3 parallelizes outermost loops with report
      =4 parallelizes every loop with report

-privatize
    Perform scalar/array privatization analysis

-range=N
    Specifies the accuracy of symbolic analysis with value ranges
      =0 disable range computation (minimal symbolic analysis)
      =1 enable local range computation (default)
      =2 enable inter-procedural computation (experimental)

-reduction=N
    Perform reduction variable analysis
      =1 enable only scalar reduction analysis (default)
      =2 enable array reduction analysis and transformation

--------------------------------------------------------------------------------
TRANSFORM
--------------------------------------------------------------------------------
-induction
    Perform induction variable substitution

-normalize-loops
    Normalize for loops so they begin at 0 and have a step of 1


-normalize-return-stmt
    Normalize return statements for all procedures

-profile-loops=N
    Inserts loop-profiling calls
      =1 every loop          =2 outermost loop
      =3 every omp parallel  =4 outermost omp parallel
      =5 every omp for       =6 outermost omp for

-teliminate-branch=N
    Eliminates unreachable branch targets
      =0 disable (default)
      =1 enable
      =2 leave old statements as comments
-tinline=mode=0|1|2|3|4:depth=0|1:pragma=0|1:debug=0|1:foronly=0|1:complement=0|
        1:functions=foo,bar,...
```

```
    (Experimental) Perform simple subroutine inline expansion transformation mode
      =0 inline inside main function (default)
      =1 inline inside selected functions provided in the "functions" sub-option
      =2 inline selected functions provided in the "functions" sub-option, when invoked
      =3 inline according to the "inlinein" pragmas
      =4 inline according to both "inlinein" and "inline" pragmasdepth
      =0 perform inlining recursively i.e. within callees (and their callees) as well
         (default)
      =1 perform 1-level inlining
   pragma
      =0 do not honor "noinlinein" and "noinline" pragmas
      =1 honor "noinlinein" and "noinline" pragmas (default)
   debug
      =0 remove inlined (and other) functions if they are no longer executed (default)
      =1 do not remove the inlined (and other) functions even if they are no longer
         executed
   foronly
      =0 try to inline all function calls depending on other options (default)
      =1 try to inline function calls inside for loops only
   complement
      =0 consider the functions provided in the command line with "functions" su
b-option (default)
      =1 consider all functions except the ones provided in the command line with
         "functions" sub-option
   functions
      =[comma-separated list] consider the provided functions.
      (Note 1: This sub-option is meaningful for modes 1 and 2 only)
      (Note 2: It is used with "complement" sub-option to determine which functions
      should be considered.)

-tsingle-call
    Transform all statements so they contain at most one function call

-tsingle-declarator
    Transform all variable declarations so they contain at most one declarator

-tsingle-return
    Transform all procedures so they have a single return statement

--------------------------------------------------------------------------------
CODEGEN
--------------------------------------------------------------------------------
-ompGen=N
    Generate OpenMP pragma
      =1 comment out existing OpenMP pragmas (default)
      =2 remove existing OpenMP pragmas
      =3 remove existing OpenMP and Cetus pragmas
      =4 keep all pragmas

-profitable-omp=N
    Inserts runtime for selecting profitable omp parallel region (See the API do
cumentation for more details)
      =0 disable
      =1 Model-based loop selection (default)
      =2 Profile-based loop selection
```

Tableau B.7 - Les différentes options du compilateur *Cetus* disponibles.

B.3 Générateur de compilateurs SableCC

SableCC (Gagnon et Hendren, 1998) est un environnement orienté-objet pour le développement de compilateurs et d'interpréteurs. Écrit en langage *Java*, cet outil transforme la spécification grammaticale en un compilateur ou un interpréteur, pour le langage décrit par cette spécification, évitant ainsi au programmeur la lourde tâche d'écrire le code des différentes parties d'un compilateur telles que l'analyseur lexical ou l'analyseur syntaxique (Agbakpem, 2006).

Pour ce faire, *SableCC* prend en entrée un fichier contenant la description du langage pour lequel le compilateur doit être écrit et produit en sortie des parties du compilateur. Le fichier d'entrée est nommé fichier de spécification du langage. À partir de ce fichier, *SableCC* génère un cadre de travail incluant un analyseur lexical et un analyseur syntaxique. Développer un compilateur ou un interpréteur avec *SableCC* demande :

- d'écrire un fichier de spécification définissant le langage pour lequel le compilateur doit être écrit;
- de fournir ce fichier à *SableCC* pour qu'il génère certaines parties du compilateur (ou de l'interpréteur);
- de compléter le code du compilateur (ou de l'interpréteur) en se servant des fichiers générés par *SableCC*.

Pour pouvoir analyser les programmes à paralléliser à travers *SableCC*, la première étape consistait à écrire une grammaire pour le langage *C*. Nous avons écrit deux grammaires pour la partie de pré-compilation (*simpleCInit* pour formater le fichier à traiter et *simpleCPreprocessing* pour ignorer les lignes de pré-compilation) et une autre grammaire pour la partie compilation (*simpleCAdder*). Les deux grammaires de pré-compilation s'assurent de la présence d'une instruction par ligne dans le ficher à traiter et ignorent toutes les commandes de pré-compilation. Par la suite, nous avons écrit une grammaire pour la partie de compilation en se basant sur le « *Committee Draft —ISO/IEC 9899:TC3* » et sur une grammaire développée par *Roger Keays* (Keays et Rakotonirainy, 2003) (voir le Tableau B.9). A travers ces grammaires, nous avons généré notre interpréteur. L'exécution de chacune des grammaires produit quatre paquets *Java* (*analysis*, *lexer*, *node* et *parser*). Notons

que l'exécution de ces trois grammaires, dans le cadre de notre projet, a généré les douze paquets suivants : *simpleCInit.analysis*, *simpleCInit.lexer*, *simpleCInit.node*, *simpleCInit.parser*, *simpleCPreprocessing.analysis*, *simpleCPreprocessing.lexer*, *simpleCPreprocessing.node*, *simpleCPreprocessing.parser*, *simpleCAdder.analysis*, *simpleCAdder.lexer*, *simpleCAdder.node* et *simpleCAdder.parser*.

Ici, nous ne présenterons que la grammaire principale de la partie de la compilation qui est la plus importante. Le Tableau B.8 contient la grammaire que nous avons écrite pour détecter les différentes parties d'un fichier écrit dans le langage de programmation *C*.

```
/* GRAMMAIRE PRINCIPALE DE LA PARTIE DE LA COMPILATION */
Package simpleCAdder ;

/* regular expressions */

Helpers
/*=============================================================================================*/
        /* sensible stuff */
        all = [0 .. 127];
        cr = 13;
        lf = 10;
        eol = cr | lf | cr lf;
        tab = 9;
        not_star = [all -'*'];
        not_star_slash = [not_star - '/'];
        spacebar = 32;
        not_spacebar = [all - spacebar];
        not_eol = [all-[13+10]];

/*=============================================================================================*/
        /* 6.4.2.1 digit, nondigit */
        digit = ['0' .. '9'];
        nondigit = ['_' + [['a' .. 'z'] + ['A' .. 'Z']]];

/*=============================================================================================*/

        /* 6.4.4.1 (various constants) */
        nonzero_digit = [digit - '0'];
        decimal_constant = nonzero_digit digit*;

        octal_digit = ['0' .. '7'];
        octal_constant = '0' octal_digit*;

        hex_prefix = '0x' | '0X';
        hex_digit = [digit + [['a' .. 'f'] + ['A' .. 'F']]];
        hex_constant = hex_prefix hex_digit*;

        unsigned_suffix = 'u' | 'U';
        long_suffix = 'l' | 'L';
        long_long_suffix = 'll' | 'LL';
        integer_suffix =
        unsigned_suffix long_suffix? |
        unsigned_suffix long_long_suffix |
        long_suffix unsigned_suffix? |
        long_long_suffix unsigned_suffix?;

        /* 6.4.4.1 (various constants) */
        integer_constant =
        decimal_constant integer_suffix? |
        octal_constant integer_suffix? |
        hex_constant integer_suffix?;
```

```
/*=========================================================================================*/

        /* 6.4.3 universal-character-name, hex-quad */
        hex_quad = hex_digit hex_digit hex_digit hex_digit;
        universal_character_name = '\u' hex_quad;

/*=========================================================================================*/

        /* 6.4.2.1 identifier-nondigit */
        identifier_nondigit = nondigit | universal_character_name;

/*=========================================================================================*/

        /* 6.4.2.2 (various constants) */
        sign = '+' | '-';
        digit_sequence = digit+;
        hex_digit_sequence = hex_digit+;
        exponent_part = ('e' | 'E') sign? digit_sequence;
        binary_exponent_part = ('p' | 'P') sign? digit_sequence;
        floating_suffix = ('f' | 'l' | 'F' | 'L');

        fractional_constant =
        digit_sequence? '.' digit_sequence |
        digit_sequence '.';

        decimal_floating_constant =
        fractional_constant exponent_part? floating_suffix? |
        digit_sequence exponent_part floating_suffix?;

        hex_fractional_constant =
        hex_digit_sequence? '.' hex_digit_sequence |
        hex_digit_sequence '.';

        hex_floating_constant =
        hex_prefix (hex_fractional_constant | hex_digit_sequence)
        binary_exponent_part floating_suffix?;

        /* 6.4.4.2 (various constants) */
        floating_constant =
        decimal_floating_constant |
        hex_floating_constant;

/*=========================================================================================*/

        /* 6.4.4.3 enumeration-constant */
        enumeration_constant_helper = identifier_nondigit
                (digit | identifier_nondigit)*;
/*=========================================================================================*

        /* 6.4.4.4 (various sequences) */
        simple_escape_seq =
                '\' ''' | '\"' | '\?' | '\\' | '\a' | '\b' | '\f' | '\n' |
                '\r' | '\t' | '\v';

        octal_escape_seq = '\' octal_digit octal_digit? octal_digit?;
        hex_escape_seq = '\x' hex_digit+;

        escape_seq =
                simple_escape_seq |
                octal_escape_seq |
                hex_escape_seq |
                universal_character_name;

        c_char = [all - [''' + ['\' + [cr + lf]]]] | escape_seq;
        c_char_seq = c_char+;
        character_constant_helper = 'L'? ''' c_char_seq ''';

/*=========================================================================================*/

        /* 6.4.5 string sequences */
        s_char = [all - ['"' + ['\' + [cr + lf]]]] | escape_seq;
        s_char_seq = s_char+;
```

```
/*=========================================================================================*/
        /* 6.4.7 header names */
        q_char = [all - ['"' + [cr + lf]]];
        q_char_seq = q_char+;
        h_char = [all - ['>' + [cr + lf]]];
        h_char_seq = h_char+;

/*=========================================================================================*/
/*
  * Tokens and keywords. The ISO standard has very few tokens: just keywords,
  * identifiers, constants, string-literals and punctuators. All the other
  * lexical elements are best expressed as Helpers as they are never referenced
  * in the Productions section.
  */

Tokens

/*=========================================================================================*/
        /* sensible stuff */
        blank = (eol | tab | ' ')+;
        comment =
                ('//' not_eol* eol?) |
                ('/*' not_star* '*'+ (not_star_slash not_star* '*'+)* '/');

/*=========================================================================================*
        /* 6.4 tokens */
/*=========================================================================================*/
        /* 6.4.1 keywords */
        kw_auto = 'auto';
        kw_break = 'break';
        kw_case = 'case';
        kw_char = 'char';
        kw_const = 'const';
        kw_continue = 'continue';
        kw_default = 'default';
        kw_do = 'do';
        kw_double = 'double';
        kw_else = 'else';
        kw_enum = 'enum';
        kw_extern = 'extern';
        kw_float = 'float';
        kw_for = 'for';
        kw_goto = 'goto';
        kw_if = 'if';
        kw_inline = 'inline';
        kw_int = 'int';
        kw_long = 'long';
        kw_register = 'register';
        kw_restrict = 'restrict';
        kw_return = 'return';
        kw_short = 'short';
        kw_signed = 'signed';
        kw_sizeof = 'sizeof';
        kw_static = 'static';
        kw_struct = 'struct';
        kw_switch = 'switch';
        kw_typedef = 'typedef';
        kw_union = 'union';
        kw_unsigned = 'unsigned';
        kw_void = 'void';
        kw_volatile = 'volatile';
        kw_while = 'while';
        kw_bool = '_Bool';
        kw_complex = '_Complex';
        kw_imaginary = '_Imaginary';

/*=========================================================================================*/

        /* 6.4.2.1 Identifiers */
        identifier = identifier_nondigit (digit | identifier_nondigit)*;
/*=========================================================================================*/
        /* 6.4.4.3 enumeration-constant */
        enumeration_constant = enumeration_constant_helper;
```

```
/*=============================================================================*/

        /* 6.4.4 constant */
        constant =
                integer_constant |
                floating_constant |
                enumeration_constant_helper |
                character_constant_helper;
        character_constant = character_constant_helper;

/*=============================================================================*/

        /* 6.4.5 string-literal */
        string_literal = 'L'? '"' s_char_seq '"';
/*=============================================================================*/

        /* 6.4.6 punctuators */
        tok_lbracket = '[';
        tok_rbracket = ']';
        tok_lpar = '(';
        tok_rpar = ')';
        tok_lbrace = '{';
        tok_rbrace = '}';
        tok_dot = '.';
        tok_arrow = '->';
        tok_plus_plus = '++';
        tok_minus_minus = '--';
        tok_amp = '&';
        tok_star = '*';
        tok_plus = '+';
        tok_minus = '-';
        tok_tilde = '~';
        tok_exclamation = '!';
        tok_slash = '/';
        tok_percent = '%';
        tok_lshift = '<<';
        tok_rshift = '>>';
        tok_lt = '<';
        tok_gt = '>';
        tok_lt_eq = '<=';
        tok_gt_eq = '>=';
        tok_eq_eq = '==';
        tok_not_eq = '!=';
        tok_caret = '^';
        tok_bar = '|';
        tok_amp_amp = '&&';
        tok_bar_bar = '||';
        tok_question = '?';
        tok_colon = ':';
        tok_semicolon = ';';
        tok_elipsis = '...';
        tok_eq = '=';
        tok_star_eq = '*=';
        tok_slash_eq = '/=';
        tok_percent_eq = '%=';
        tok_plus_eq = '+=';
        tok_minus_eq = '-=';
        tok_lshift_eq = '<<=';
        tok_rshift_eq = '>>=';
        tok_amp_eq = '&=';
        tok_caret_eq = '^=';
        tok_bar_eq = '|=';
        tok_comma = ',';
        tok_hash = '#';
        tok_hash_hash = '##';
        tok_lt_colon = '<:';
        tok_colon_gt = ':>';
        tok_lt_percent = '<%';
        tok_percent_gt = '%>';
        tok_percent_colon = '%:';
        tok_percent_colon_percent_colon = '%:%:';
/*=============================================================================*/
Ignored Tokens
        comment,
```

```
        blank;
/*=============================================================================================*/
/* Concrete Syntax Tree */
Productions

        /* 6.9 translation-unit */
        translation_unit = external_declaration+ ;

        /* 6.9 external-declaration */
        external_declaration =
                {defn} function_definition |
                {decl} declaration;

        /* 6.9.1 function-definition */
        function_definition =
                declaration_specifiers declarator declaration_list? compound_statement;

        /* 6.9.1 declaration-list */
        declaration_list = declaration+;

/*=============================================================================================*/
        /* 6.5.1 primary-expression */
        primary_expression =
                {identifier} identifier |
                {constant} constant |
                {string} string_literal |
                {expression} tok_lpar expression tok_rpar;

        /* 6.5.2 postfix-expression */
        postfix_expression =
                {primary} primary_expression |
                {bracket} postfix_expression tok_lbracket expression tok_rbracket |
                {par} postfix_expression tok_lpar argument_expression_list? tok_rpar |
                {dot} postfix_expression tok_dot identifier |
                {arrow} postfix_expression tok_arrow identifier |
                {plus_plus} postfix_expression tok_plus_plus |
                {minus_minus} postfix_expression tok_minus_minus |
                {cast1} tok_lpar type_name tok_rpar tok_lbrace initializer_list
                                tok_rbrace |
                {cast2} tok_lpar type_name tok_rpar tok_lbrace initializer_list
                                tok_comma tok_rbrace;

        /* 6.5.2 argument-expression-list */
        argument_expression_list =
                {single} assignment_expression |
                {list} argument_expression_list tok_comma assignment_expression;

        /* 6.5.3 unary-expression */
        unary_expression =
                {postfix} postfix_expression |
                {plus_plus} tok_plus_plus unary_expression |
                {minus_minus} tok_minus_minus unary_expression |
                {cast} unary_operator cast_expression |
                {sizeof1} kw_sizeof unary_expression |
                {sizeof2} kw_sizeof tok_lpar type_name tok_rpar;

        /* 6.5.3 unary-operator */
        unary_operator =
                {amp} tok_amp |
                {star} tok_star |
                {plus} tok_plus |
                {minus} tok_minus |
                {tilde} tok_tilde |
                {exclamation} tok_exclamation;

        /* 6.5.4 cast-expression */
        cast_expression =
                {no} unary_expression |
                {cast} tok_lpar type_name tok_rpar cast_expression;

        /* 6.5.5 multiplicative-expression */
        multiplicative_expression =
                {no} cast_expression |
                {mult} multiplicative_expression tok_star cast_expression |
```

```
        {divide} multiplicative_expression tok_slash cast_expression |
        {mod} multiplicative_expression tok_percent cast_expression;

/* 6.5.6 additive-expression */
additive_expression =
        {no} multiplicative_expression |
        {plus} additive_expression tok_plus multiplicative_expression |
        {minus} additive_expression tok_minus multiplicative_expression;

/* 6.5.7 shift-expression */
shift_expression =
        {no} additive_expression |
        {lshift} shift_expression tok_lshift additive_expression |
        {rshift} shift_expression tok_rshift additive_expression;

/* 6.5.8 relational-expression */
relational_expression =
        {no} shift_expression |
        {lt} relational_expression tok_lt shift_expression |
        {gt} relational_expression tok_gt shift_expression |
        {lt_eq} relational_expression tok_lt_eq shift_expression |
        {gt_eq} relational_expression tok_gt_eq shift_expression;

/* 6.5.9 equality-expression */
equality_expression =
        {no} relational_expression |
        {eq_eq} equality_expression tok_eq_eq relational_expression |
        {not_eq} equality_expression tok_not_eq relational_expression;

/* 6.5.10 AND-expression */
and_expression =
        {no} equality_expression |
        {and} and_expression tok_amp equality_expression;

/* 6.5.11 exclusive-OR-expression */
exclusive_or_expression =
        {no} and_expression |
        {xor} exclusive_or_expression tok_caret and_expression;

/* 6.5.12 inclusive-OR-expression */
inclusive_or_expression =
        {no} exclusive_or_expression |
        {or} inclusive_or_expression tok_bar exclusive_or_expression;

/* 6.5.13 logical-AND-expression */
logical_and_expression =
        {no} inclusive_or_expression |
        {and} logical_and_expression tok_amp_amp inclusive_or_expression;
/* 6.5.14 logical-OR-expression */
logical_or_expression =
        {no} logical_and_expression |
        {or} logical_or_expression tok_bar_bar logical_and_expression;


/* 6.5.15 conditional-expression */
conditional_expression =
        {no} logical_or_expression |
        {cond} logical_or_expression tok_question expression tok_colon
                        conditional_expression ;

/* 6.5.16 assignment-expression */
assignment_expression =
        {no} conditional_expression |
        {assign} unary_expression assignment_operator assignment_expression;

/* 6.5.16 assignment-operator */
assignment_operator =
        {eq} tok_eq |
        {star_eq} tok_star_eq |
        {slash_eq} tok_slash_eq |
        {percent_eq} tok_percent_eq |
        {plus_eq} tok_plus_eq |
        {minus_eq} tok_minus_eq |
        {lshift_eq} tok_lshift_eq |
```

```
                {rshift_eq} tok_rshift_eq |
                {amp_eq} tok_amp_eq |
                {caret_eq} tok_caret_eq |
                {bar_eq} tok_bar_eq;

        /* 6.5.17 expression */
        expression =
                {no} assignment_expression |
                {list} expression tok_comma assignment_expression;
/*=============================================================================*/

        /* 6.6 constant-expression */
        constant_expression = conditional_expression;

/*=============================================================================*/

        /* 6.7 declaration */
        declaration = declaration_specifiers init_declarator_list? tok_semicolon ;

        /* 6.7 declaration-specifiers */
        declaration_specifiers =
                {storage} storage_class_specifier declaration_specifiers? |
                {type_spec} type_specifier declaration_specifiers? |
                {type_qual} type_qualifier declaration_specifiers? |
                {function} function_specifier declaration_specifiers? ;

        /* 6.7 init-declarator-list */
        init_declarator_list =
                {single} init_declarator |
                {list} init_declarator_list tok_comma init_declarator;

        /* 6.7 init-declarator */
        init_declarator =
                {plain} declarator |
                {assign} declarator tok_eq initializer;

/*=============================================================================*/

        /* 6.7.1 storage-class-specifier */
        storage_class_specifier =
                {typedef} kw_typedef |
                {extern} kw_extern |
                {static} kw_static |
                {auto} kw_auto |
                {register} kw_register;
/*=============================================================================*/

        /* 6.7.2 type-specifier */
        type_specifier =
                {void} kw_void |
                {char} kw_char |
                {short} kw_short |
                {int} kw_int |
                {long} kw_long |
                {float} kw_float |
                {double} kw_double |
                {signed} kw_signed |
                {unsigned} kw_unsigned |
                {bool} kw_bool |
                {complex} kw_complex |
                {imaginary} kw_imaginary |
                {struct} struct_or_union_specifier |
                {enum} enum_specifier ;

        /* 6.7.2.1 struct-or-union-specifier */
        struct_or_union_specifier =
                {defined} struct_or_union identifier? tok_lbrace struct_declaration_list
                                tok_rbrace |
                {not_defined} struct_or_union identifier;

        /* 6.7.2.1 struct-or-union */
        struct_or_union =
                {struct} kw_struct |
                {union} kw_union;
```

```
        /* 6.7.2.1 struct-declaration-list */
        struct_declaration_list = struct_declaration+ ;

        /* 6.7.2.1 struct-declaration */
        struct_declaration =
                specifier_qualifier_list struct_declarator_list tok_semicolon ;

        /* 6.7.2.1 specifier-qualifier-list */
        specifier_qualifier_list =
                {spec_first} type_specifier specifier_qualifier_list? |
                {qual_first} type_qualifier specifier_qualifier_list? ;
        /* 6.7.2.1 struct-declarator-list */
        struct_declarator_list =
                {single} struct_declarator |
                {list} struct_declarator_list tok_comma struct_declarator;

        /* 6.7.2.1 struct-declarator */
        struct_declarator =
                {plain} declarator |
                {with_const} declarator? tok_colon constant_expression;

        /* 6.7.2.2 enum-specifier */
        enum_specifier =
                {values1} kw_enum identifier? tok_lbrace enumerator_list tok_rbrace |
                {values2} kw_enum identifier? tok_lbrace enumerator_list tok_comma
                                tok_rbrace |
                {novalues} kw_enum identifier;

        /* 6.7.2.2 enumerator-list */
        enumerator_list =
                {single} enumerator |
                {list} enumerator_list tok_comma enumerator;

        /* 6.7.2.2 enumerator */
        enumerator =
                {plain} enumeration_constant |
                {assign} enumeration_constant tok_eq constant_expression;

/*=============================================================================================*/

        /* 6.7.3 type-qualifier */
        type_qualifier =
                {const} kw_const |
                {restrict} kw_restrict |
                {volatile} kw_volatile;

/*=============================================================================================*/

        /* 6.7.4 function-specifier */
        function_specifier = kw_inline;

/*=============================================================================================*/
        /* 6.7.5 declarator */
        declarator = pointer? direct_declarator;

        /* 6.7.5 direct-declarator */
        direct_declarator =
                {ident} identifier |
                {par} tok_lpar declarator tok_rpar |
                {qual1} direct_declarator tok_lbracket type_qualifier_list?
                                assignment_expression? tok_rbracket |
                {qual2} direct_declarator tok_lbracket kw_static type_qualifier_list?
                                assignment_expression tok_rbracket |
                {qual3} direct_declarator tok_lbracket type_qualifier_list kw_static
                                assignment_expression tok_rbracket |
                {qual4} direct_declarator tok_lbracket type_qualifier_list?tok_star
                                tok_rbracket |
                {param_types} direct_declarator tok_lpar parameter_type_list tok_rpar |
                {ident_list} direct_declarator tok_lpar identifier_list? tok_rpar;
        /* 6.7.5 pointer */
        pointer =
                {single} tok_star type_qualifier_list? |
                {recursive} tok_star type_qualifier_list? pointer;
```

```
        /* 6.7.5 type-qualifier-list */
        type_qualifier_list = type_qualifier+ ;

        /* 6.7.5 parameter-type-list */
        parameter_type_list =
                {plain} parameter_list |
                {elipsis} parameter_list tok_comma tok_elipsis;
        /* 6.7.5 parameter-list */
        parameter_list =
                {single} parameter_declaration |
                {list} parameter_list tok_comma parameter_declaration;

        /* 6.7.5 parameter-declaration */
        parameter_declaration =
                {plain} declaration_specifiers declarator |
                {abstract} declaration_specifiers abstract_declarator?;

        /* 6.7.5 identifier-list */
        identifier_list =
                {single} identifier |
                {list} identifier_list tok_comma identifier;

/*=============================================================================================*/

        /* 6.7.6 type-name */
        type_name = specifier_qualifier_list abstract_declarator?;

        /* 6.7.6 abstract-declarator */
        abstract_declarator =
                {pointer} pointer |
                {direct} pointer? direct_abstract_declarator;

        /* 6.7.6 direct-abstract-declarator */
        direct_abstract_declarator =
                {par} tok_lpar abstract_declarator tok_rpar |
                {bracket1} direct_abstract_declarator? tok_lbracket
                                        assignment_expression? tok_rbracket |
                {bracket2} direct_abstract_declarator? tok_lbracket tok_star
                                        tok_rbracket |
                {params} direct_abstract_declarator? tok_lpar parameter_type_list?
                                tok_rpar;

/*=============================================================================================*/

        /* 6.7.7 typedef-name */
        typedef_name = identifier;

/*=============================================================================================*/

        /* 6.7.8 initializer */
        initializer =
                {assign} assignment_expression |
                {list1} tok_lbrace initializer_list tok_rbrace |
                {list2} tok_lbrace initializer_list tok_comma tok_rbrace;

        /* 6.7.8 initializer-list */
        initializer_list =
                {single} designation? initializer |
                {list} initializer_list tok_comma designation? initializer;
        /* 6.7.8 designation */
        designation = designator_list tok_eq;
        /* 6.7.8 designator-list */
        designator_list = designator+ ;

        /* 6.7.8 designator */
        designator =
                {brackets} tok_lbracket constant_expression tok_rbracket |
                {dot} tok_dot identifier;

/*=============================================================================================*/

        /* 6.8 statement */
        statement =
                {labeled} labeled_statement |
```

```
        {compound} compound_statement |
        {expression} expression_statement |
        {selection} selection_statement |
        {iteration} iteration_statement |
        {jump} jump_statement;

/* 6.8.1 labeled-statement */
labeled_statement =
        {plain} identifier tok_colon statement |
        {case} kw_case constant_expression tok_colon statement |
        {default} kw_default tok_colon statement ;

/* 6.8.2 compound-statement */
compound_statement = {alt} tok_lbrace block_item_list? tok_rbrace;

/* 6.8.2 block-item-list */
block_item_list = block_item+;

/* 6.8.2 block-item */
block_item =
        {decl} declaration |
        {statement} statement;

/* 6.8.3 expression-statement */
expression_statement = {exp} expression? tok_semicolon;

/* 6.8.4 selection-statment */
selection_statement =
        {if} kw_if tok_lpar expression tok_rpar statement |
        {ifelse} kw_if tok_lpar expression tok_rpar compound_statement kw_else
                [other]:compound_statement |
        {switch} kw_switch tok_lpar expression tok_rpar statement;
/* 6.8.5 iteration-statement */
iteration_statement =
        {while} kw_while tok_lpar expression tok_rpar statement |
        {do} kw_do statement kw_while tok_lpar expression tok_rpar
                tok_semicolon |
        {for1} kw_for tok_lpar [decl]:expression? [a]:tok_semicolon

                [cond]:expression?  [b]:tok_semicolon [iter]:expression?
                tok_rpar statement |
        {for2} kw_for tok_lpar declaration [cond]:expression? tok_semicolon
                [iter]:expression? tok_rpar statement ;

/* 6.8.6 jump-statement */
jump_statement =
        {goto} kw_goto identifier tok_semicolon |
        {continue} kw_continue tok_semicolon |
        {break} kw_break tok_semicolon |
        {return} kw_return expression? tok_semicolon;
```

Tableau B.8 - Grammaire écrite pour la partie de la compilation (*simpleCAdder*) de *SableCC*.

B.4 Méthode supplémentaire de division des tâches des boucles entre les différents processeurs

Outre l'utilisation de l'opérateur *modulo*, la parallélisation des boucles peut s'effectuer à travers le calcul du nombre de tours de la boucle à paralléliser. Une méthode de calcul du nombre de tours effectués par chaque processeur peut être définie de la manière suivante (voir le Tableau B.9) :

```
DÉBUT
...
Si (operateurBoucle == (≥ ou ≤)) alors
        tailleBoucle = ((indiceFin – IndiceDebut) + 1) / pasIcrementation;
Sinon Si (operateurBoucle == (< ou >)) alors
        tailleBoucle = ((indiceFin – IndiceDebut) + 1) / pasIncrementation;

nbToursParProcesseur = tailleBoucle / nbProceseurs;
debut = (nbToursParProcesseur * processeurId) + indiceDebut;
fin = MINIMUM (nbToursParProcesseur * (processeurId + 1), indiceFin);

Pour (i = debut; i < fin; i++) {
        Code exécuté en parallèle;
}
...
FIN
```

Figure B.1 - Méthode de calcul du nombre de tours de la boucle à paralléliser.

ANNEXE C

SCRIPTS ET PROGRAMMES DÉVELOPPÉS POUR LA DÉTECTION DE TRANSFERTS HORIZONTAUX DE GÈNES COMPLETS ET PARTIELS

L'annexe C présente quelques exemples du code source développé pour la détection de transferts complets et partiels. On retrouve ici les scripts écrits en langage *Perl* (*run_hgt_partiels.pl*, *run_hgt_complets.pl*), qui appellent les différentes versions du programme de détection de transferts horizontaux de gènes écrites en langages *C/C++*. Notons que le script *run_hgt_complets.pl* inclut la version standard, la version interactive, la version consensus, ainsi que la version parallèle du programme *HGT-Detection*. Le script *run_hgtp_partiels.pl*, quant à lui, inclut seulement la version de détection des HGTs partiels (deuxième algorithme de détections des THGs partiels présenté dans la thèse). Puis, nous présentons la fonction principale du programme *HGT-Detection* (*hgt.cpp*), ainsi que de la version parallèle de la fonction de recherche des transferts *findBestHGTTab*(). Nous terminons cette annexe par la présentation des programmes de génération d'arbres phylogénétiques aléatoires utilisés dans nos simulations.

C.1 Script Perl : *run_hgt_partiels.pl*

Ce script permet l'exécution du programme de détection de THGs *partiels* à partir des données récupérées sur le site Web de *T-Rex*. Ce script prend en entrée les paramètres suivants : l'arbre d'espèces, les séquences de l'arbre de gène, la méthode d'inférence d'arbre, la taille de la fenêtre coulissante, le déplacement de la fenêtre, le type de séquences à traiter, le nombre d'arbres pour le bootstrap, le seuil de bootstrap minimum acceptable et le nombre

de fenêtres consécutives. Ce script fournit en sortie : le fichier contenant les transferts détectés et le fichier des statistiques sur les transferts détectés.

Exemple de la ligne de commande avec toutes les options disponibles :

```
perl usagers/run_hgt_partiels.pl -speciesTreeFile=[Species tree file]
-geneSequencesFile=[Gene sequences file] -opt_m=[PhyML|NJ] -opt_ws=[100] -opt_ss=[10]
-opt_st=[DNA|RNA|AA] -opt_nr=[10] -opt_bm=[60] -opt_ncw=[3]\n";
```

Le code source du script *run_hgt_partiels.pl* est présenté dans le Tableau C.1.

```
#!/usr/bin/perl
use strict;
use Devel::Profiler;
use warnings;

print STDOUT "\n\n";
print STDOUT "================================================================\n";
print STDOUT "| Partial HGT-DETECTION V.1.1 (August, 2011) by Alix Boc and
Vladimir Makarenkov |\n";
print STDOUT "================================================================\n";

#=========================================================
#= VÉRIFICATION DES ARGUMENTS DE LA LIGNE DE COMMANDE
#=========================================================
if( scalar @ARGV < 1){
    print "Erreur\nusage : $0 speciesTreeFile=[Species tree file]
        geneSequencesFile=[Gene sequences file] opt_m=[PhyML|NJ]
        opt_ws=[100] opt_ss=[10] opt_st=[DNA|RNA|AA] opt_nr=[10]
        opt_bm=[60] opt_ncw=[3]\n";
    print "\nspeciesTreeFile\tSpecies tree file name. The species tree should be
        in the Newick format";
        print "\ngeneSequencesFile\tGene sequences file name. The sequences
        should be in the Phylip format";
        print "\nopt_m \tTree inference (PhyML or NJ)
        : default NJ";
        print "\nopt_ws\tSliding window size
        : default=100";
    print "\nopt_ss\tStep size
        : default=10";
    print "\nopt_st\tSequence type (DNA,AA)
        : default=AA";
        print "\nopt_nr\tNumber of replicates in bootstrap
        : default=10";
    print "\nopt_bm\tDisplay transfers with bootstrap support higher than x%
        : default=60%";
    print "\nopt_ncw\tNumber of consecutive windows necessary to validate a
        transfer: default=3";
    print "\n";
    exit 0;
}
#=============================================================================
#= Gestion des paramètres = valeurs par defaut
#=============================================================================
my $fenetre                     = 100;  #= taille de la fenetre
my $increment                   = 10;   #= deplacement de la fenetre
my $nbBootstrap          = 10;   #= nombre d'arbre pour le bootstrap
my $boot_min                    = 60;   #= bootstrap minimum acceptable
my $pos_debut_globale           = 0;    #= position de debut
my $reconstruction              = "NJ"; #= NJ,ML
my $sequence_type               = "AA"; #= DNA,AA
my $speciesTreeFile             = "";
my $geneSequencesFile           = "";
my $min_hgt_length              = 0;
my $ncw                         = 3;
my $command_line                = "";
my $cmd                  = "";
```

```
my @myarray;

#============================================================================
#= Gestion des paramètres = nouvelles valeurs
#============================================================================
my $param;
my @tab_tmp;

for(my $i=0;$i<scalar @ARGV;$i++){
        $param =  $ARGV[$i];
        chomp($param);
        @tab_tmp = split("=",$param);
        if($tab_tmp[0] eq "opt_st"){
                $sequence_type = $tab_tmp[1];
                if(($sequence_type ne "DNA") && ($sequence_type ne "AA")){
                        print STDOUT "$sequence_type : Unknown sequence
                           type (DNA,AA)\n";
                        exit;
                }
        }
        elsif($tab_tmp[0] eq "opt_ws"){
                $fenetre = $tab_tmp[1];
                $min_hgt_length = $fenetre;
        }
        elsif($tab_tmp[0] eq "opt_ncw"){
                $ncw = $tab_tmp[1];
        }
        elsif($tab_tmp[0] eq "opt_m"){
                $reconstruction = $tab_tmp[1];
        }
        elsif($tab_tmp[0] eq "opt_ss"){
                $increment = $tab_tmp[1];
        }
        elsif($tab_tmp[0] eq "opt_bm"){
                $boot_min = $tab_tmp[1];
        }
        elsif($tab_tmp[0] eq "opt_nr"){
                $nbBootstrap = $tab_tmp[1];
        }
        elsif($tab_tmp[0] eq "speciesTreeFile"){
                $speciesTreeFile = $tab_tmp[1];
        }
        elsif($tab_tmp[0] eq "geneSequencesFile"){
                $geneSequencesFile = $tab_tmp[1];
        }
        else{
                print STDOUT $tab_tmp[0] . ": Unknown parameters\n";
                exit;
        }
}

$min_hgt_length = $fenetre + ($ncw-1)*$increment;
#============================================================
#= Affichage des données de simulation
#============================================================
print STDOUT "Species Tree filename   = $speciesTreeFile\n";
print STDOUT "Gene Sequences filename = $geneSequencesFile\n";
print STDOUT "Sliding windows size    = $fenetre\n";
print STDOUT "Step size               = $increment\n";
print STDOUT "Number of replicates    = $nbBootstrap\n";
print STDOUT "Minimum Bootstrap       = $boot_min\n";
print STDOUT "Tree reconstruction     = $reconstruction\n";
print STDOUT "Sequence Type           = $sequence_type\n";
print STDOUT "Number of consecutive windows necessary
    to validate a transfer = $ncw\n";
open (OUT,">phgt_output.txt");
print OUT "=================================================\n";
print OUT "| Partial HGT-DETECTION V.1.1 (August, 2011) by Alix
Boc and Vladimir Makarenkov |\n";
print OUT "=================================================\n";
print OUT "Species Tree filename   = $speciesTreeFile\n";
print OUT "Gene Sequences filename = $geneSequencesFile\n";
print OUT "Sliding windows size    = $fenetre\n";
print OUT "Step size               = $increment\n";
```

```
print OUT "Number of replicates     = $nbBootstrap\n";
print OUT "Minimum Bootstrap        = $boot_min\n";
print OUT "Tree reconstruction      = $reconstruction\n";
print OUT "Sequence Type            = $sequence_type\n";
print OUT "Number of consecutive windows necessary to validate a transfer = $ncw\n";
close (OUT);

my $phyml                = "exec/phyml";
my $seqboot              = "exec/seqboot";
my $neighbor             = "exec/neighbor";
my $rf                   = "exec/rf";
my $dnadist      = "exec/dnadist";
my $protdist     = "exec/protdist";
my $consense             = "exec/consense";

#============================================================
#= PARTIAL HGT-DETECTION
#============================================================
my $speciesTreeNewick;
my %geneSequences        = ();
my $taille_sequence      = 0;
my $nombre_especes       = 0;
my $tmp;
my $hgtFoundFile         = "hgtdetectes.txt";
my $infile               = "infile";
my $outfile              = "outfile";
my $outtree              = "outtree";
my $intree             = "intree";
my $inputHgtFile         = "input.txt";
my $outputHgtFile        = "output.txt";
my $seqbootConf  = "seqbootConf.txt"; #= fichier de config de seqboot
my $dnadistConf  = "dnadistConf.txt"; #= fichier de config de dnadist
my $protdistConf         = "protdistConf.txt";#= fichier de config de protdist
my $neighborConf         = "neighborConf.txt";#= fichier de config de neighbor
my $phymlConf            = "phymlConf.txt";    #= fichier de config de phyml
my $seed                 = 3;                  #= seed pour les progs Phylip
my $log          = "log.txt";         #= fichier log
my $pos_fin;
my %transferts   = ();
my %transferts2  = ();
my $pos_debut            = $pos_debut_globale;
my $bootTree             = 0;
my $RF_distance  =-1;
my $bootTreeFile = "bootTreeFile.txt";
my $consenseConf = "consenseConf.txt";
my $nbNan                = 0;

if( -e "$outputHgtFile.tmp"){
    $command_line = "rm -rf $outputHgtFile.tmp";executeLinuxCommandOutLoop();
}

#=========================================================
#= LECTURE DES DONNÉES (arbre et sÉquences)
#=========================================================
@tab_tmp = file_get_contents($speciesTreeFile);
$speciesTreeNewick = $tab_tmp[0];

%geneSequences = chargerSequence($geneSequencesFile);
$nombre_especes = scalar keys(%geneSequences);
@tab_tmp = keys(%geneSequences);
$taille_sequence = length($geneSequences{$tab_tmp[0]});

#=========================================================
#= INITIALISATION DES FICHIERS DE CONFIGURATION
#=========================================================
$command_line = "echo \"R\n$nbBootstrap\ny\n$seed\n\" > $seqbootConf";
executeLinuxCommandOutLoop();
$command_line = "echo \"M\nd\n$nbBootstrap\ny\n\" > $dnadistConf";
executeLinuxCommandOutLoop();
$command_line = "echo \"M\n$nbBootstrap\n$seed\ny\n\" > $neighborConf";
executeLinuxCommandOutLoop();
$command_line = "echo \"M\nD\n$nbBootstrap\np\np\np\ny\n\" > $protdistConf";
executeLinuxCommandOutLoop();
$command_line = "echo \"y\n\" > $consenseConf";
```

```
executeLinuxCommandOutLoop();

if($sequence_type eq "DNA"){
        $command_line = "echo \"infile\nb\n$nbBootstrap\ny\ny\n\" > $phymlConf";
        executeLinuxCommandOutLoop();
        $command_line = "echo \"$infile\n+\n+\n0\nL\n+\nb\n$nbBootstrap\ny\
           y\n\" > $phymlConf";
        executeLinuxCommandOutLoop();
}
else{
        $command_line = "echo \"$infile\nd\n+\nM\nM\nM\n+\n0\nL\n+
           \nb\n$nbBootstrap\ny\nY\n\"> $phymlConf";
        executeLinuxCommandOutLoop();
}

print STDOUT "\n=================================================================";
print STDOUT "\n|  Interval   |   RF   | Bootstrap | Detected transfers";
print STDOUT "\n=================================================================";
my $last = 0;
my $zone = 0;
my $nouvelle_zone = "yes";
my $iteration = 0;

while ( ($pos_debut < $taille_sequence) && ($last == 0)){
        if(($pos_debut + $fenetre) > ($taille_sequence-1)){
                $pos_debut = $taille_sequence - 1 - $fenetre;
                $last = 1;
        }
        printf (STDOUT "\n| [%3d - %3d]",$pos_debut,$pos_debut + $fenetre);

        $pos_fin = $pos_debut + $fenetre;
        #=========================================================
        #= CRÉATION DU FICHIER INPUT DE SÉQUENCES
        #=========================================================
        open(OUT,">$infile") || die "Cannot open $infile ($!)";
        print OUT "$nombre_especes $fenetre";
        foreach my $elt( keys(%geneSequences)){
                my $espace;
                for(my $i=1;$i<=(10-length($elt));$i++){
                        $espace .= " ";
                }
                print OUT "\n" . $elt ."$espace". substr($geneSequences{$elt},
                   $pos_debut,$fenetre);
        }
        close(OUT);

        #--execute("cat $speciesTreeFile > $inputHgtFile");
        $command_line = "cat $speciesTreeFile > $inputHgtFile";
        executeLinuxCommandInLoop();

        #=========================================================
        #= GÉNÉRATION DES ARBRES
        #=========================================================
        if($reconstruction eq "NJ"){
                #--execute("rm -rf $outfile >> $log");
                $command_line = "rm -rf $outfile >> $log";
                executeLinuxCommandInLoop();

              #= generation des autres arbres
                #--execute("$seqboot < $seqbootConf >> $log");
                $command_line = "$seqboot < $seqbootConf >> $log";
                executeSeqBoot();

                #--execute("cat $outfile > $infile");#= ajout des autres arbres
                $command_line = "cat $outfile > $infile";
                executeLinuxCommandInLoop();

                #--execute("rm -rf $outfile >> $log");
                $command_line = "rm -rf $outfile >> $log";
                executeLinuxCommandInLoop();
                if($sequence_type eq "DNA"){
                     #= calcul des matrices de distances
                        #--execute("$dnadist < $dnadistConf >> $log");
                        $command_line = "$dnadist < $dnadistConf >> $log";
```

```
                executeDnadist();
                if(!-e $outfile){
                        print STDOUT "\nProblems with dnadist...\n";
                        exit;
                }
                #--execute("mv $outfile $infile");
                $command_line = "mv $outfile $infile";
                executeLinuxCommandInLoop();
                #--execute("rm -rf $outtree >> $log");
                $command_line = "rm -rf $outtree >> $log";
                executeLinuxCommandInLoop();
        }
        else{
               #= calcul des matrices de distances
                #--execute("$protdist < $protdistConf >> $log")
                $command_line = "protdist < $protdistConf >> $log";
                executeProtdist();

                $command_line = "mv $outfile $infile";
                executeLinuxCommandInLoop();
                $command_line = "rm -rf $outtree >> $log";
                executeLinuxCommandInLoop();
        }
        $nbNan = `grep "nan" $infile | wc -l`;
        if($nbNan == 0){
        $command_line = "$neighbor < $neighborConf >> $log";
        executeNeighbor();
        $command_line = "cp $outtree $intree";
        executeLinuxCommandInLoop();
        $command_line = "echo \"$speciesTreeNewick\" > $inputHgtFile";
        executeLinuxCommandInLoop();
        $command_line = "tr -d '\n\r' < $outtree | sed
         's/;/;\\\n/g' | sed 's/-//g' >> $inputHgtFile";
        executeLinuxCommandInLoop();
        $command_line = "rm -rf $outtree $outfile";
        executeLinuxCommandInLoop();
        $command_line = "$consense < $consenseConf >> $log";
        executeConsense();
        $command_line = "grep 'Sets in' -A 100 $outfile |
           grep 'Sets NOT' -B 100 | grep '\*' |
           grep \"[0-9][0-9]*.[0-9][0-9]*\" -o > $bootTreeFile";
        executeLinuxCommandInLoop();
        my @tab_tmp=file_get_contents("$bootTreeFile");
        $bootTree=0;
        for(my $i=0;$i<scalar @tab_tmp;$i++){
                $bootTree += ($tab_tmp[$i]*100)/($nbBootstrap+1);
        }
        $bootTree /= scalar @tab_tmp;
        @tab_tmp = file_get_contents2("$inputHgtFile");
        my $arbre2=$tab_tmp[1];
        $RF_distance = Robinson_and_Foulds($speciesTreeNewick,$arbre2);
        printf( STDOUT " |  %3d  |    %3.0lf%%   | ",
           $RF_distance,$bootTree);
        }
        else{
          printf( STDOUT " |  Distance matrices cannot be build");
        }
}

else{
        if($reconstruction eq "PhyML"){
                $command_line = "rm -rf $infile" . "_*";
                executeLinuxCommandInLoop();
                $command_line = "$phyml < $phymlConf >> $log";
                executePhyML();
                my @tab_tmp = file_get_contents("$infile" . "_phyml_tree.txt");
                my $chaine = $tab_tmp[0];
                (my @tab_boot) = ($chaine =~ /\)([0-9][0-9]*):/g);
                my $total = 0;
                my $boot_cpt=0;
                foreach my $val (@tab_boot){
                        $total += $val;
                        $boot_cpt++;
                }
```

```
                    $bootTree = (($total/$boot_cpt)*100)/$nbBootstrap;
                    $chaine =~ s/\)[0-9][0-9]*:/\):/g;
                    @tab_tmp = file_get_contents("$speciesTreeFile");
                    $RF_distance = Robinson_and_Foulds($tab_tmp[0],$chaine);
                    printf( STDOUT " |  %3d  |    %3.0lf%%   |",
                       $RF_distance,$bootTree);
                    $command_line = "echo \"\n$chaine\" >> $inputHgtFile";
                    executeLinuxCommandInLoop();
                    @tab_tmp = file_get_contents("$infile" .
                       "_phyml_boot_trees.txt");

                    foreach $chaine (@tab_tmp){
                            chomp ($chaine);
                            $command_line = "echo \"\n$chaine\"
                              >> $inputHgtFile";
                            executeLinuxCommandInLoop();
                    }
            }
    }
    #=======================================================
    #= DÉTECTION DE TRANSFERTS AVEC BOOTSTRAP
    #=======================================================
    if(($RF_distance > 1) && ($bootTree >= 0) && ($nbNan == 0 )){
            if($nouvelle_zone eq "yes"){
                    $zone++;
            }
            $nouvelle_zone = "no";
            $iteration++;
            $command_line = "rm -rf output.txt outputWeb.txt results.txt
               nomorehgt.txt log_hgt.txt return.txt";
            executeLinuxCommandInLoop();
            $command_line = "perl run_hgt.pl -inputfile=$inputHgtFile
               -bootstrap=yes >> $log";
            executeHGT_Detection();
            $command_line = "cat $inputHgtFile >> $log";
            executeLinuxCommandInLoop();
            $command_line = "cat inputfileformated.txt >> $log";
            executeLinuxCommandInLoop();
            $command_line = "cat log_hgt.txt >> $log";
            executeLinuxCommandInLoop();
            $command_line = "cat $outputHgtFile >> $log";
            executeLinuxCommandInLoop();

            #=======================================================
            #= LECTURE DES RÉSULTATS
            #=======================================================
            my @tab_output = file_get_contents($outputHgtFile);
            open(OUT,">>$outputHgtFile.tmp") || die "Cannot open
              $outputHgtFile.tmp($!)";
            my @tab_list_dest=();
            my @tmp_tab;
            open(IN,"$outputHgtFile") || die "Cannot open
              $outputHgtFile ($!)";
            foreach my $ligne (@tab_output){
                    chomp($ligne);
                    $ligne =~ s/ //g;
                    print STDOUT "*";
                    (my $source,my $dest,my $bootstrap) = split("<>",$ligne);
                    @tmp_tab    = split(",",$source);
                    $source     = join(",",@tmp_tab);
                    my @tab_source = split(",",$source);
                    @tmp_tab    = split(",",$dest);
                    $dest       = join(",",@tmp_tab);
                    my @tab_dest  = split(",",$dest);
                    if($bootstrap > $boot_min){
                            print OUT "$iteration<>$RF_distance<>
                           $pos_debut<>$pos_fin<>$source<>
                           $dest<>$bootstrap\n";
                    }
            }
            close(IN);
            close(OUT);
    }
    else{
```

```
                $nouvelle_zone = "yes";
        }
        if($RF_distance < 2){
                $pos_debut += 2*$increment;
        }
        if(($pos_debut + $fenetre) == ($taille_sequence-1)){
                $pos_debut = $taille_sequence;
        }

        $pos_debut = $pos_debut + $increment;
}
print STDOUT "\n=================================================================\n";
open (OUT,">>phgt_output.txt");
print OUT "\n\nPartial HGT detected with bootstrap support higher than $boot_min%:\n";
print OUT "=================================================================";
{
        my $max_rf = -1;
        $command_line = "cat $outputHgtFile.tmp  > $outputHgtFile";
        executeLinuxCommandOutLoop();
        open(IN,"$outputHgtFile") || die "Cannot open $outputHgtFile ($!)";
        while(my $ligne = <IN>){
                chomp($ligne);
                if($ligne =~ /<>/){
                        $ligne =~ s/ //g;
                        (my $iteration,my $RF,$pos_debut,$pos_fin,my $source,my
                           $dest,my $bootstrap) = split("<>",$ligne);
                         if($max_rf == -1){
                                $max_rf = $RF;
                        }
                        {
                        my @tmp_tab = split(",",$source);
                        $source = join(",",@tmp_tab);
                        @tmp_tab = split(",",$dest);
                        $dest = join(",",@tmp_tab);
                        if($bootstrap > $boot_min){
                                my @tmp_tab2 = split("<>",$ligne);
                                print OUT "\n\nTransfer  : " .
                                 $tmp_tab2[4] ."->". $tmp_tab2[5];
                                printf( OUT "\nBootstrap : %1.1lf%%",$tmp_tab2[6]);
                                print OUT "\nInterval  : " . $tmp_tab2[2]
                                   . "-" . $tmp_tab2[3];
                                if(exists($transferts{"$source->$dest"} )){
                                        for(my $i=$pos_debut;$i<=$pos_fin;$i++){
                                                $transferts{"$source->$dest"}[$i] = 1;
                                        }
                                }
                                else{
                                        for(my $i=0;$i<$taille_sequence;$i++){
                                                $transferts{"$source->$dest"}[$i] = 0;
                                        }
                                        for(my $i=$pos_debut;$i<=$pos_fin;$i++){
                                                $transferts{"$source->$dest"}[$i] = 1;
                                        }
                                }
                        }
                }
        }
        close(IN);
}

#=====================================================
#= AFFICHAGE DES RÉSULTATS
#=====================================================
print OUT "\n\n\nOverlapping partial HGT detected with bootstrap
higher than $boot_min% and validated over $ncw consecutive windows\n";
print OUT "=========================================================================";
my %results  = ();
my %compteur = ();
my %interval_debut = ();
my %interval_fin = ();
open (IN,$outputHgtFile)||die "Impossible d'ouvrir le fichier $outputHgtFile($!)";

while( my $ligne=<IN>){
        $ligne =~ s/\[//g;
```

```
        $ligne =~ s/\]//g;
        my @tmp = split("<>",$ligne);
        my $key = $tmp[4] . "->" . $tmp[5];
        if(exists($results{$key})){
                if($tmp[6] >= ($boot_min/2)){
                        if( ($tmp[2] <= $interval_fin{$key} ) && ($tmp[2] >
                       $interval_debut{$key} ) ){
                                $interval_fin{$key} = $tmp[3];
                        }
                        if( ($tmp[2] > $interval_fin{$key} ) &&
                       (($tmp[3]-$interval_fin{$key}) < 40) ){
                                $interval_fin{$key} = $tmp[3];
                        }
                        $results{$key}  = $results{$key} + $tmp[6];
                        $compteur{$key} = $compteur{$key} + 1;
                }
        }
        else{
                $results{$key}  = $tmp[6];
                $compteur{$key} = 1;
                $interval_debut{$key}  = $tmp[2];
                $interval_fin{$key}  = $tmp[3];
        }

}

foreach my $key (keys %results){
        if(($results{$key}/$compteur{$key} > $boot_min) &&
        ( ($interval_fin{$key}-$interval_debut{$key})>= $min_hgt_length)){
                print OUT "\n\nTransfer          : $key";
                printf( OUT "\nAverage bootstrap : %1.1lf%%",
                   $results{$key}/$compteur{$key});
                print OUT "\nInterval          : " . $interval_debut{$key}
                    ."-". $interval_fin{$key};
        }
}
print OUT "\n\n";
close (OUT);

open(OUT,">$hgtFoundFile") || die "impossible d'ouvrir $hgtFoundFile ($!)";
my $saut = "";

my $pred = 0;

foreach my $elt (keys %transferts){
        for(my $i=0;$i<$taille_sequence;$i++){
                if(($transferts{$elt}[$i] == 1) && ($pred == 0)){
                        $saut = "\n";
                        $pred = 1;
                }
                elsif((($transferts{$elt}[$i] == 0)||($i == ($taille_sequence-1)))
               && ($pred == 1)){
                        $pred=0;
                }
        }
}
close(OUT);
clean();
#--execute("mv phgt_output.txt output.txt");
$command_line = "mv phgt_output.txt output.txt";
executeLinuxCommandOutLoop();
print STDOUT "\nSee results in the file output.txt\n\nEnd of the computation ....\n";
#=======================================
#= SOUS-PROGRAMMES
#=======================================

sub delete_elt_array{
        (my $elt,my @tab) = @_;
        my @new_tab=();
        foreach my $val (@tab){
                push(@new_tab,$val) if($val ne $elt);
        }
        return @new_tab;
}
```

```
sub est_detecte{
        my $trans_det = $_[0]."->".$_[1];
        my $fichier= $_[2];
        #=============================================
        #= LECTURE DES TRANSFERTS SIMULÉS
        #=============================================
        open (IN,$fichier) || die "Impossible d'ouvrir le fichier $fichier ($!)";
        my @hgt_simules = <IN>;
        close(IN);
        for(my $i=0;$i<scalar @hgt_simules;$i++){
                my $lignel = $hgt_simules[$i];
                chomp($lignel);
                (my $trans_sim,my $sd,my $sf) = split("<>",$lignel);
                if($trans_sim eq $trans_det){
                        return 1;
                }
        }
        return 0;
}

sub Robinson_and_Foulds{
        file_put_contents("rf_input.txt",$_[0] . "\n" . $_[1]);
        $command_line = "$rf rf_input.txt rf_output.txt rf_tmp.txt
           rf_output.tmp rf_matrices.txt";
        executeLinuxCommandInLoop();
        my @tab_tmp = file_get_contents("rf_output.txt");
        (my $RF) = ($tab_tmp[5] =~ /.*=\ ([0-9][0-9]*)/);
        return $RF;
}

sub file_get_contents{
        open(IN,$_[0]) || die "Impossible d'ouvrir " . $_[0] . " ($!)";
        my $i=0;
        my @return = <IN>;
        close(IN);
        return @return;
}

sub file_get_contents2{
        open(IN,$_[0]) || die "Impossible d'ouvrir " . $_[0] . " ($!)";
        my $i=0;
        my @return = ();
        while(my $ligne =<IN>){
           if( $ligne =~ ";"){
               $return[$i] = $ligne;
               $i++;
           }
        }
        close(IN);
        return @return;
}

sub file_put_contents{
        my $fichier = $_[0];
        open(OUT,">$fichier") || die "Impossible d'ouvrir $fichier ($!)";
        print OUT $_[1];
        close(OUT);
}

# Routine d'exécution d'une commande
sub execute{
        my $cmd = $_[0];
        my $retour = system("$cmd");
}

# Routune d'écriture dans un fichier
sub writeInFile{
        (my $contenu, my $fichier) = @_;
        open (OUT,">$fichier") || die "impossible d'ouvrir $fichier ($!)";
        print OUT "$contenu";
        close (OUT);
}
```

```
# Routine de chargement d'une séquence
sub chargerSequence{
        my $file = $_[0];
        my $tmp=0;
        my %geneSequences = ();
        open(IN,$file) || die "Cannot open $file ($!)";
        while (my $ligne = <IN>){
                        chomp($ligne);
                        if($tmp > 0){
                                (my $name, my $sequence) = split(" ",$ligne);
                                $geneSequences{$name} = $sequence;
                        }
                        $tmp ++;
        }
        close (IN);
        return %geneSequences;
}

#Routine pour supprimer les fichiers créés lors de l'exécution du script
sub clean
{
    $command_line = "rm -rf bootTreeFile.txt infile_phyml_boot_stats.txt input.txt
        nomorehgt.txt phymlConf.txt rf_input.txt speciesRootLeaves.txt tmp_input.txt";
    executeLinuxCommandOutLoop();
    $command_line = "rm -rf consenseConf.txt geneRootLeaves.txt infile_phyml_boot_trees.txt
        output.txt protdistConf.txt rf_output.tmp speciesRootTmp.txt";
    executeLinuxCommandOutLoop();
    $command_line = "rm -rf geneRootTmp.txt infile_phyml_stats.txt intree output.txt.tmp
        rf_output.txt speciesRoot.txt";
    executeLinuxCommandOutLoop();
    $command_line = "rm -rf geneRoot.txt infile_phyml_tree.txt outputWeb.txt
        results2.txt rf_tmp.txt";
    executeLinuxCommandOutLoop();
    $command_line = "rm -rf dnadistConf.txt hgtdetectes.txt inputfileformated.txt
        results.txt sync_projet.sh";
    executeLinuxCommandOutLoop();
    $command_line = "rm -rf errorFile.txt infile input_.txt neighborConf.txt return.txt
        seqbootConf.txt test.txt outtree outfile";
    executeLinuxCommandOutLoop();
}
#============================================================================
#== Alpha Boubacar Diallo - Profiling
#== Note: Division of the excute code for each section
#============================================================================
sub executeDnadist {
        system("$command_line");
}
sub executeProtdist {
        system("$command_line");
}

sub executeNeighbor {
        system("$command_line");
}

sub executeConsense {
        system("$command_line");
}

sub executePhyML {
        system("$command_line");
}

sub executeHGT_Detection {
        system("$command_line");
}

sub executeSeqBoot{
        system("$command_line");
}

sub executeLinuxCommandOutLoop{
        system("$command_line");
}
```

```
sub executeLinuxCommandInLoop{
        system("$command_line");
}
```

Tableau C.1 - Code source du script Perl *run_hgt_partiels.pl*.

C.2 Script Perl : *run_hgt_complets.pl*

Ce script permet l'exécution des différentes versions du programme de détection de THGs *complets* à partir des données récupérées du site Web de *T-Rex*. Ce script prend en entrée les paramètres suivants : l'arbre d'espèces, l'arbre de gène, ainsi que différentes options sur le processus de détection de transferts horizontaux de gènes complets telles que le critère d'optimisation, la version du programme de détection, le seuil minimal du bootstrap ou du consensus. Il fournit en sortie : le fichier contenant les transferts détectés et le fichier des statistiques sur les transferts détectés.

Exemple de la ligne de commande avec toutes les options disponibles :

```
usr/bin/perl usagers/run_hgt_complets.pl -bootstrap=[yes|no]-consensus=[yes|no]
-consvalue=[0..100] -nbhgt=$maxHGT -scenario=[unique|multiple] -stepbystep=[yes|no]
-mode=$mode -speciesroot=$rootSpeciesVal -generoot=$rootGeneVal -version=web -path=$path
-inputfile=input.txt -outputfile=output.txt -subtree=$subTreeVal -criterion[MC|RF|DB]
-version=[parallel|sequentiel]
```

Le code source du script est présenté dans le Tableau C.2.

```
#!/usr/bin/perl
use strict;
use warnings;
use File::Copy;
#=======================================================
#= VÉRIFICATION DES ARGUMENTS DE LA LIGNE DE COMMANDE
#=======================================================
if( scalar @ARGV < 0){
    print "\nErreur\nusage : $0";
    exit 0;
}
my $cmd = "hgt3.4 ";      # Commande de la version séquentielle
my $cmd_par = "";         # Commande de la version parallèle
my $inputfile = "";
my $outputfile="output.txt";
my $bootstrap = "no";
my $consensus = "no";
my $cons=50;
my $path = "./"; #"./";
my $viewtree="no";
my @tmp_tab;
my @tmp_tab_init;
my %hgt;
my $ligne;
my $nbLines = 5;
my %hgt_number_tab;
my %hgt_description_tab;
my %hgt_compteur_tab;
my %hgt_compteur_tab_reverse;
my %hgt_criterion_tab;
my %hgt_criterion_tab2;
my %hgt_nbHGT_tab;
my @hgt_pos;
my @hgt_pos2;
my $total_hgt;      #= Total de hgts
my $total_trivial;  #= Total de hgts triviaux
```

```
my $val_retour=0;   #= nombre de hgt trouve
my @hgt_tab;
my $rand_bootstrap = 0;
my $speciesroot = "midpoint";
my $generoot = "midpoint";
my $stepbystep = "no";
my $mode;
my $version ="sequentiel";        #= Version du programme à exécuter
# Lecture des paramètres du script
foreach my $elt (@ARGV){
    $cmd .= $elt . " " if(($elt !~ "consensus")&&($elt !~ "consvalue"));
    if($elt =~ "bootstrap"){
        @tmp_tab = split("=",$elt);
        $bootstrap = $tmp_tab[1];
        chomp($bootstrap);
    }
   if($elt =~ "consensus"){
        @tmp_tab = split("=",$elt);
        $consensus = $tmp_tab[1];
        chomp($consensus);
    }
    if($elt =~ "consvalue"){
        @tmp_tab = split("=",$elt);
        $cons = $tmp_tab[1];
        chomp($cons);
    }
    if($elt =~ "speciesroot"){
        @tmp_tab = split("=",$elt);
        $speciesroot = $tmp_tab[1];
        chomp($speciesroot);
    }
    if($elt =~ "generoot"){
        @tmp_tab = split("=",$elt);
        $generoot = $tmp_tab[1];
        chomp($generoot);
    }
    if($elt =~ "inputfile"){
        @tmp_tab = split("=",$elt);
        $inputfile = $tmp_tab[1];
    }
    if($elt =~ "path"){
        @tmp_tab = split("=",$elt);
        $path = $tmp_tab[1];
    }
    if($elt =~ "viewtree"){
        @tmp_tab = split("=",$elt);
        $viewtree = $tmp_tab[1];
    }
    if($elt =~ "outputfile"){
        @tmp_tab = split("=",$elt);
        $outputfile = $tmp_tab[1];
    }
    if($elt =~ "help"){
                print_description();
                print_help();
        exit;
    }
    if($elt =~ "stepbystep"){
        @tmp_tab = split("=",$elt);
        $stepbystep = $tmp_tab[1];
    }
    if($elt =~ "version"){
        @tmp_tab = split("=",$elt);
        $version = $tmp_tab[1];
    }
}

if($consensus eq "yes" && $bootstrap eq "yes"){
    $bootstrap="no";
}
if($stepbystep eq "yes"){
    $version = "sequentiel";
}
```

```
# Définition des variables du script
my $inputfile_bkp = $inputfile;
$inputfile = "$path" . "$inputfile";
$outputfile = "$path" . "$outputfile";
$cmd_par =  $cmd;
$cmd   = "usagers/" . $cmd;
my $results = "$path" . "results.txt";
my $firstResults = "$path" . "firstResults.txt";
my $results2   = "$path" . "results2.txt";
my $firstResults2   = "$path" . "firstResults2.txt";
my $tmp_input = "$path" . "tmp_input.txt";
my $input_no_space    = "$path" . "input_no_space.txt";
my $return_file = "$path" . "return.txt";
my $log_file = "$path" . "log.txt";
my $log_file1 = "$path" . "log1.txt";
my $output_tmp;
my $outputWeb = "$path" . "outputWeb.txt";
my $firstOutputWeb = "$path" . "firstOutputWeb.txt";
my $outputConsensus = "$path" . "outputCons.txt";
my $outputWebConsensus = "$path" . "outputWebCons.txt";
my $generootfile = "$path" . "geneRootLeaves.txt";
my $generootfileBranches = "$path" . "geneRoot.txt";
my $speciesrootfile = "$path" . "speciesRootLeaves.txt";
my $speciesrootfileBranches = "$path" . "speciesRoot.txt";
my $generootfiletmp = "$path" . "geneRootTmp.txt";
my $speciesrootfiletmp = "$path" . "speciesRootTmp.txt";
my $inputfileformated = "$path" . "inputfileformated.txt";
my $prehgtfile = "$path" . "prehgt.txt";
my $hgt = "";
my $premier = 1;
my $nb = 1;
my $nbHGT=0;
my $nbHGT2=0;

 #===== PRINT HEADER =====
print_title();

#= linux like
`rm -rf $results $outputfile $log_file $return_file`;

#= windows
#`del $results $outputfile $log_file $return_file $outputWeb`;

#===== CHECKING FILES =====
if( $inputfile eq ""){
    print STDOUT "\n\nRUN_HGT : There is no input file";
    exit -1;
}
if( ! -e $inputfile){
    print STDOUT "\n\nRUN_HGT : $inputfile doesn't exist";
    exit -1;
}

if( ($speciesroot eq "file") &&
( ! -e $speciesrootfile && ! -e $speciesrootfileBranches) ){
    print STDOUT "\n\nRUN_HGT : $speciesrootfile doesn't exist";
    exit -1;
}
if( ($generoot eq "file") && ( ! -e $generootfile
&& ! -e $generootfileBranches) ){
    print STDOUT "\n\nRUN_HGT : $generootfile doesn't exist";
    exit -1;
}
#=== LECTURE DE L'ARBRE D'ESPECES ===
open(IN,"$inputfile") ||  die "Cannot open $inputfile";
open(OUT,">$inputfileformated") ||  die "Cannot open $inputfileformated";
while($ligne = <IN>){
    chomp($ligne);
    $ligne =~ s/;/;\n/g;
    if($ligne ne ""){
        print OUT $ligne;
    }
}
close(IN);
```

```
close(OUT);
open(IN,"$inputfileformated") ||  die "Cannot open $inputfileformated";
my @trees_tab = <IN>;
close(IN);

#============================================================================
#======================== EXÉCUTION DU PROGRAMME ============================
#============================================================================
$cmd .= "-inputfile=tmp_input.txt -outputfile=output.txt"; # > $log_file";
$cmd_par .= "-inputfile=tmp_input.txt -outputfile=output.txt";

my $nbTrees = 0 ; # scalar @trees_tab - 1;

#== The program need at least 2 trees
if((scalar @trees_tab < 2)){
    exit_program(-1,$return_file,"PERL : nombre d'arbres invalide");
}
print_minidoc();
for (my $i=0;(($i< scalar @trees_tab) && $trees_tab[$i] =~ ";");)
{
    open(IN,">$tmp_input") ||  die "Cannot open $tmp_input";
    #============================================================================
    #= In the bootstrap case, we need to change the speciesroot and
    #= generoot option for "file" from the first replicate.
    #============================================================================
    if($bootstrap eq "yes" || $consensus eq "yes" ){
        if($i == 0){
            print IN $trees_tab[0] . $trees_tab[1];
            $i=2;
            #$cmd .= " -randbootstrap=$rand_bootstrap";
        }
        else{
            print IN $trees_tab[0] . $trees_tab[$i++];
        }
        if($i > 2){
            if($cmd !~ /printWeb=no/){
                $cmd .= " -printWeb=no ";
                $cmd_par .= " -printWeb=no ";
            }
        }
        if($nbTrees == 1){
            $cmd =~ s/-generoot=[a-z][a-z]* //;
            $cmd =~ s/-speciesroot=[a-z][a-z]* //;
            if($consensus eq "yes"){
                $cmd .= " -generoot=midpoint -speciesroot=midpoint";
                $cmd_par .= " -generoot=midpoint -speciesroot=midpoint";
            }
            else{
                $cmd .= " -generoot=file -speciesroot=file";
                $cmd_par .= " -generoot=file -speciesroot=file";
            }

            #= linux
            `cp $generootfile $generootfiletmp`;
            `cp $speciesrootfile $speciesrootfiletmp`;
            #= windows
            # `copy $generootfile $generootfiletmp`;
            # `copy $speciesrootfile $speciesrootfiletmp`;
        }

        if($nbTrees > 1){
            #== linux like
            `cp $generootfiletmp $generootfile`;
            `cp $speciesrootfiletmp $speciesrootfile`;
            #== windows
            # `copy $generootfiletmp $generootfile`;
            # `copy $speciesrootfiletmp $speciesrootfile`;
        }
        if($rand_bootstrap == 1){
            $rand_bootstrap = 0;
            $cmd =~ s/randbootstrap=1/randbootstrap=0/;
        }
        else{
            $rand_bootstrap = 1;
```

```
            $cmd =~ s/randbootstrap=0/randbootstrap=1/;
        }
    }
    else{
        print IN $trees_tab[$i++] . $trees_tab[$i++];
    }
    close(IN);

    #== traitement des cas de transferts avant detection (en mode interactif)
    if($stepbystep eq "yes"){
        if(-e $outputWeb){
            my @prehgt;
            my @prehgt_valide;
            my @prehgt_sequence;
            open(PREHGT,"$outputWeb") ||
                die "Probleme avec l'ouverture du fichier $outputWeb";
            my @tab_prehgt = <PREHGT>;
            foreach my $ligne( @tab_prehgt){
                chomp($ligne);
                if($ligne =~ /^hgt_reels =/){
                    $ligne =~ s/ = /,/;
                    @prehgt = split(",",$ligne);
                }
                if($ligne =~ /^hgt_valide =/){
                    $ligne =~ s/ = /,/;
                    @prehgt_valide = split(",",$ligne);
                }
                if($ligne =~ /^hgt_sequence/){
                    $ligne =~ s/ = /,/;
                    @prehgt_sequence = split(",",$ligne);
                }
            }
            close(PREHGT);
            print STDOUT join("-",@prehgt) . "\n";
            print STDOUT join("-",@prehgt_valide) . "\n";
            print STDOUT join("-",@prehgt_sequence) . "\n";
            open(PREHGT,">$prehgtfile")|| die "Impossible d'ouvrir $prehgtfile";
            print STDOUT "\nnbHGT  = " . $#prehgt_valide . "\n\n";
            for(my $i=1;$i< scalar(@prehgt_valide);$i++){
            print PREHGT $prehgt_sequence[$i] . " ". $prehgt[4*$i-3] .  " " .
                $prehgt[4*$i-2] . " " . $prehgt[4*$i-1]." " .$prehgt[4*$i-0]
                . " " . $prehgt_valide[$i] . "\n";
            }
            close(PREHGT);
        }
    }
    print STDOUT "\nComputation " . ++$nbTrees . " in progress...";
    if($consensus eq "yes"){
        $cmd =~ s/bootstrap=no/bootstrap=yes/;
    }

    # Exécution du programme de détection de transferts
    $cmd_par = "/usr/bin/perl /home/trex/public_html/hgt_par.pl  ". $cmd_par;
    if($version eq "parallel"){
        execute_hgt("$cmd_par >> $log_file");
    }

    else{
        execute_hgt("$cmd >> $log_file");
    }
    if ($consensus eq "yes"){
        open(INFO, $outputWeb);   # Open the file
        my @lines = <INFO>;               # Read it into an array
        close(INFO);                      # Close the file
        foreach my $line( @lines){
            if ($line =~ /hgt = /){
                if ($premier ==1){
                    $premier = 0;
                }
                else{
                    $line=~ s/hgt = /,/;
                }
                $line =~ s/\s+$//;
                $hgt = $hgt . $line;
```

```
        }
    }
}
if( ! -e $results){
    print STDOUT "\n\nRUN_HGT : An error has occured during computation.
        Check the log file ($log_file) for more details ! ";
    exit -1;
}
print STDOUT "formatting results...";
if((($i == 2) && ($viewtree eq "yes")) || (($i == 1) && ($viewtree eq "yes"))){
   exit_program(0,$return_file,"RUN_HGT : We just want to see the input tress");
}

#=============================================================================
#======================== LECTURE DES RÉSULTATS ==============================
#=============================================================================
open(IN,"$results") ||  die "\nCannot open $results ! !!";
@hgt_tab = <IN>;
close(IN);
exit_program(-1,$return_file,"RUN_HGT:result file empty")if(scalar @hgt_tab==0);
$mode = $hgt_tab[1];
chomp($mode);

# Traitement des résultats du premier arbre de gène
if(  ($i == 2) || (($bootstrap eq "no") && ($consensus eq "no"))  ){
     print STDOUT "done";
    $nbHGT=0;
    $nbHGT2=0;
    for(my $j=2;$j<= scalar @hgt_tab;){
        if($hgt_tab[$j] =~ /^[0-9]/){
            my $cpt = read_line($hgt_tab[$j++]);
            for(my $k=0;$k<$cpt;$k++){
                my $hgt_number = read_line($hgt_tab[$j++]);
                my $source_list = read_line($hgt_tab[$j++]);
                my $dest_list = read_line($hgt_tab[$j++]);
                my $transfer_description2 = read_line($hgt_tab[$j++]);
                my $transfer_description = "From subtree($source_list)
                      to subtree ($dest_list)";
                my $criterion_list = read_line($hgt_tab[$j++]);
                my $criterion_list2 = read_line($hgt_tab[$j++]);
                $hgt_number_tab{"$source_list -> $dest_list"}= $hgt_number;
                $hgt_description_tab{"$source_list -> $dest_list"} =
                $transfer_description;
                $hgt_compteur_tab{"$source_list -> $dest_list"} = 1;
                $hgt_criterion_tab{"$source_list -> $dest_list"}=$criterion_list;
                $hgt_criterion_tab2{"$source_list -> $dest_list"}= $criterion_list2;
                $hgt_nbHGT_tab{"$source_list -> $dest_list"}  = $cpt;
                $hgt_pos[$nbHGT++] = "$source_list -> $dest_list";
            }
            $hgt_pos2[$nbHGT2++] = read_line($hgt_tab[$j++]);
        }
        else{
             @tmp_tab_init = split(",",$hgt_tab[0]);
             @tmp_tab = split(" ",$hgt_tab[$j]);
            $total_hgt = $tmp_tab[1];
            $total_trivial = $tmp_tab[2];
            if( ($bootstrap eq "no") && ($consensus eq "no") ){
                open(OUT,">>$outputfile") || die "Cannot open $outputfile";
                if($total_hgt > 0){
                    print_result();
                }
                else{
                    print OUT " : no HGTs have been found !";
                }
                close(OUT);
                exit_program($val_retour,$return_file,"PERL : pas de bootstrap,
                    on traite un seul input");
                %hgt_number_tab=();
                %hgt_description_tab=();
                %hgt_compteur_tab=();
                %hgt_criterion_tab=();
                %hgt_criterion_tab2=();
                %hgt_nbHGT_tab=();
                @hgt_pos=();
```

```
                @hgt_pos2=();
            }
            $j = (scalar @hgt_tab) + 1;
        }
    }
    copy($results, $firstResults) or die "File cannot be copied.";
    copy($outputWeb, $firstOutputWeb) or die "File cannot be copied.";
    copy($results2, $firstResults2) or die "File cannot be copied.";
}
# Traitement des résultats des arbres de gènes restants
if((($bootstrap eq "yes") || ($consensus eq "yes"))   && ($i > 2)){
    print STDOUT "done";
    if($bootstrap eq "yes") {
        $nbHGT=0;
        $nbHGT2=0;
    }
    for(my $j=2;$j<= scalar @hgt_tab;){
        if($hgt_tab[$j] =~ /^[0-9]/){
            my $cpt = read_line($hgt_tab[$j++]);
            for(my $k=0;$k<$cpt;$k++){
                my $hgt_number = read_line($hgt_tab[$j++]);
                my $source_list = read_line($hgt_tab[$j++]);
                my $dest_list = read_line($hgt_tab[$j++]);
                my $transfer_description2 = read_line($hgt_tab[$j++]);
                my $transfer_description = "From subtree ($source_list) to
                subtree ($dest_list)";
                my $criterion_list = read_line($hgt_tab[$j++]);
                my $criterion_list2 = read_line($hgt_tab[$j++]);
                if(exists $hgt_compteur_tab{"$source_list -> $dest_list"}){
                    $hgt_compteur_tab{"$source_list -> $dest_list"}   += 1;
                }
                else{
                    if($consensus eq "yes"){
                        $hgt_number_tab{"$source_list -> $dest_list"}=
                        $hgt_number;
                        $hgt_description_tab{"$source_list -> $dest_list"} =
                            $transfer_description;
                        $hgt_compteur_tab{"$source_list -> $dest_list"}= 1;
                        $hgt_criterion_tab{"$source_list -> $dest_list"} =
                            $criterion_list;
                        $hgt_criterion_tab2{"$source_list -> $dest_list"} =
                            $criterion_list2;
                        $hgt_nbHGT_tab{"$source_list -> $dest_list"} = $cpt;
                        $hgt_pos[$nbHGT++] = "$source_list -> $dest_list";
                    }
                }
            }
            if($consensus eq "yes"){
                $hgt_pos2[$nbHGT2++] = read_line($hgt_tab[$j++]);
             }

             else{
                $j++;
            }
        }
        else{
            $j = (scalar @hgt_tab) + 1;
        }
    }
    if($consensus eq "yes")
    {
        $total_hgt =  $nbHGT;
    }
    print STDOUT "\nLe nombre de transferts" . $nbHGT ;
}
}

# On recopie les premiers rÃ©sultats sauvegardÃ©s.
copy($firstResults, $results) or die "File cannot be copied.";
copy($firstOutputWeb, $outputWeb) or die "File cannot be copied.";
copy($firstResults2, $results2) or die "File cannot be copied.";
unlink($firstResults);
unlink($firstOutputWeb);
unlink($firstResults2);
```

```
if($version eq "sequentiel"){
    # Affichage dans le cas de version consensus
    if($consensus eq "yes"){
        open(OUT,">>$outputfile" ) || die "Cannot open $outputfile";
        print_result_consensus();
        close(OUT);
    }

    # Affichage dans le cas du boostrap
    if($bootstrap eq "yes"){
        open(OUT,">>$outputfile") || die "Cannot open $outputfile";
        print_result();
        close(OUT);
    }

    # Affichage dans la version interactive
    if($stepbystep eq "yes"){
        open(OUT,">>$outputfile") || die "Cannot open $outputfile";
        print_result();
        close(OUT);
    }
}
else{
    open(OUT,">>$outputfile") || die "Cannot open $outputfile";
    print_result_par();
    close(OUT);
}

exit_program($val_retour,$return_file,"PERL : fin normale du programme");

#================================================================================
#================================ FONCTIONS =====================================
#================================================================================
# Routine pour lire une ligne
sub read_line{
    my ($line) = @_;
    chomp($line);
    return $line;
}

# Routine terminer le programme
sub exit_program{
    my($val,$file,$message) = @_;
    open(RET,">$file") || die "Cannot open $file";
    print RET $val;
    close(RET);
    print STDOUT "\n";
    exit;
}

# Routine pour exécuter le programme de détection de transferts
sub execute_hgt{
    my ($cmd) = @_;
    my $retour = 0;
    $retour= system($cmd);
}

# Routine pour afficher les résultats de la version standard et interactive
sub print_result
{
    my $sim_hgt = "yes";
    open (SIM , ">res_sim.txt") if ($sim_hgt eq "yes");
    my $nbHGT2=0;
    my $newGroup=0;
    my $cpt=1;
    my @tmp_tab = split(",",$hgt_tab[0]);
    print OUT    "============================================================\n";
    print OUT    "| Program : HGT Detection 3.4 - March, 2012                |\n";
    print OUT    "| Authors   : Alix Boc, Alpha Boubabcar Diallo and Vladimir
        Makarenkov             |\n";
    print OUT    "|             (Universite du Quebec a Montreal)            |\n";
    print OUT    "| This program computes a unique scenario of horizontal gene
        transfers (HGT) for |\n";
```

```
print OUT   "| the given pair of species and gene phylogenetic trees.   |\n";
print OUT    "=============================================================\n";
print OUT "\nSpecies tree :\n". $trees_tab[0] . "\nGene Tree :\n" .
    $trees_tab[1];
print OUT "\n\n=============================================";
print OUT "\n= Criteria values before the computation ";
print OUT "\n=============================================";
if($bootstrap eq "yes"){
    printf (OUT "\nRobinson and Foulds distance (RF)=%d",$tmp_tab_init[0]);
    printf (OUT "\nLeast-squares coefficient(LS) = %1.3lf",$tmp_tab_init[1]);
    printf (OUT "\nBipartition dissimilarity  = %1.1lf\n",$tmp_tab_init[2]);
}
else{
    printf (OUT "\nRobinson and Foulds distance (RF)= %d",$tmp_tab[0]);
    printf (OUT "\nLeast-squares coefficient(LS) = %1.3lf",$tmp_tab[1]);
    printf (OUT "\nBipartition dissimilarity   = %1.1lf\n",$tmp_tab[2]);
}
printf(OUT "\n\nBootstrap values were computed with %d gene trees",$nbTrees)
    if($bootstrap eq "yes");
print OUT "\n\n";
foreach my $elt( @hgt_pos){
    if($newGroup == 0 ) {
        print OUT "\n=====================================================";
        if($hgt_nbHGT_tab{"$elt"} == 1){
            print OUT "\n| Iteration #$cpt : ". $hgt_nbHGT_tab{"$elt"} .
            " HGT was found";
        }
        else{
          print OUT "\n| Iteration #$cpt : ". $hgt_nbHGT_tab{"$elt"} .
            " HGTs were found";
        }
        print OUT "\n=====================================================";
        print OUT "\n|";
        $newGroup = 1;
        $cpt++;
    }
    print OUT "\n| "  . $hgt_number_tab{"$elt"};
    my @tmp = split(" -> ",$elt);
    my $elt_reverse = $tmp[1] . " -> " . $tmp[0];
    if(($bootstrap eq "yes")) { #&&($hgt_number_tab{"$elt"} !~ "Trivial")){
        printf(OUT "(bootstrap value = %3.1lf%% inverse = %3.1lf%%) ",
            $hgt_compteur_tab{$elt}*100/$nbTrees,$hgt_compteur_tab_reverse
            {$elt_reverse}*100/$nbTrees);
        my @tmp_sim_tab = split("->",$elt);
        printf(SIM "%s<>%s<>%3.1lf\n",$tmp_sim_tab[0],$tmp_sim_tab[1],
        $hgt_compteur_tab{$elt}*100/$nbTrees) if ($sim_hgt eq "yes");
    }
    print OUT "\n| "  . $hgt_description_tab{"$elt"};
    print OUT "\n| "  . $hgt_criterion_tab{"$elt"};
    print OUT "\n| "  . $hgt_criterion_tab2{"$elt"};
    if($mode eq 'mode=monocheck'){
        print OUT "\n=====================================================\n";
    }

    else{
        print OUT "\n| ";
    }
    my $tmp = "HGT " .$hgt_nbHGT_tab{"$elt"}." / ".$hgt_nbHGT_tab{"$elt"}." ";
    my $tmp2 = $tmp . " Trivial";

   if(( $tmp =~ $hgt_number_tab{"$elt"}) ||($tmp2 =~ $hgt_number_tab{"$elt"})){
        if ($mode eq 'mode=multicheck') {
            print OUT "\n=================================================";
            print OUT "\n|After this iteration the criteria values are as
               follows :";
            print OUT "\n| " . $hgt_pos2[$nbHGT2++] ;
            print OUT "\n=================================================\n";
        }
        $newGroup=0;
    }
}
print OUT "\nTotal number of HGTs : $total_hgt ";
print OUT "(". ($total_hgt-$total_trivial) ." regular + " . $total_trivial .
    " trivial HGTs)" if( $total_trivial > 0);
```

```
    $val_retour = $total_hgt;
    open(OUTWEB,">>$outputWeb");
    if($bootstrap eq "yes"){
        print OUTWEB "\nbootHGT = ";
        my $first=0;
        foreach my $elt( @hgt_pos){
            if(($bootstrap eq "yes")&&($hgt_number_tab{"$elt"} !~ "Trivial")){
                if($first==0){
                    printf(OUTWEB "%3.0lf",$hgt_compteur_tab{$elt}*100/$nbTrees);
                }
                else{
                    printf(OUTWEB ",%3.0lf",$hgt_compteur_tab{$elt}*100/$nbTrees);
                }
                $first=1;
            }
            elsif($bootstrap eq "yes"){
                if($first==0){
                    printf OUTWEB "0";
                }
                else{
                    printf OUTWEB ",0";
                }
                $first=1;
            }
        }
    }
    close OUTWEB;
    close (SIM) if ($sim_hgt eq "yes");
}

# Routine pour compater deux transferts
sub par_elt { return $hgt_compteur_tab{"$b"}  <=> $hgt_compteur_tab{"$a"} }

# Routine pour afficher les résultats de la version parallèle
sub print_result_par{
    my $nbHGT2=0;
    my $newGroup=0;
    my $cpt=1;
    my @tmp_tab = split(",",$hgt_tab[0]);
    print OUT    "============================================================\n";
    print OUT    "| Program : HGT Detection 3.3 - March, 2011                |\n";
    print OUT    "| Authors   : Alix Boc, Alpha Boubabcar Diallo and Vladimir
        Makarenkov            |\n";
    print OUT    "|             (Universite du Quebec a Montreal)            |\n";
    print OUT    "| This program computes a unique scenario of horizontal
        gene transfers (HGT) for |\n";
    print OUT    "| the given pair of species and gene phylogenetic trees.  |\n";
    print OUT    "============================================================\n";

    print OUT "\nSpecies tree :\n". $trees_tab[0] . "\nGene Tree :\n" . $trees_tab[1];
    print OUT "\n\n=============================================";
    print OUT "\n= Criteria values before the computation ";
          print OUT "\n=============================================";
    if($bootstrap eq "yes"){
        printf (OUT "\nRobinson and Foulds distance (RF)= %d",$tmp_tab_init[0]);
        printf (OUT "\nLeast-squares coefficient(LS) = %1.3lf",$tmp_tab_init[1]);
        printf (OUT "\nBipartition dissimilarity  = %1.1lf\n",$tmp_tab_init[2]);
    }
    else{
        printf (OUT "\nRobinson and Foulds distance (RF) = %d",$tmp_tab[0]);
        printf (OUT "\nLeast-squares coefficient(LS) = %1.3lf",$tmp_tab[1]);
        printf (OUT "\nBipartition dissimilarity  = %1.1lf\n",$tmp_tab[2]);
    }
    printf(OUT "\n\nBootstrap values were computed with %d gene trees",$nbTrees)
        if($bootstrap eq "yes");
    print OUT "\n\n";
    foreach my $elt( @hgt_pos){
        if($newGroup == 0 ) {
            print OUT "\n=====================================================";
            if($hgt_nbHGT_tab{"$elt"} == 1){
                print OUT "\n| Iteration #$cpt : ". $hgt_nbHGT_tab{"$elt"} .
                    " HGT was found";
            }
            else{
```

```
                print OUT "\n| Iteration #$cpt : ". $hgt_nbHGT_tab{"$elt"}
                    ." HGTs were found";
            }
            print OUT "\n=======================================================";
            print OUT "\n|";
            $newGroup = 1;
             $cpt++;
        }
        print OUT "\n| "  . $hgt_number_tab{"$elt"};
        printf(OUT "(bootstrap value = %3.11f%%) ",$hgt_compteur_tab{$elt}*100/
            $nbTrees,$hgt_compteur_tab{$elt}, $nbTrees) if(($bootstrap eq "yes")
            &&($hgt_number_tab{"$elt"} !~ "Trivial"));
        print OUT "\n| "  . $hgt_description_tab{"$elt"};
        print OUT "\n| "  . $hgt_criterion_tab{"$elt"};
        print OUT "\n| "  . $hgt_criterion_tab2{"$elt"};
        if($mode eq 'mode=monocheck'){
            print OUT "\n=======================================================\n";
        }
        else{
            print OUT "\n| ";
        }
        my $tmp = "HGT ".$hgt_nbHGT_tab{"$elt"}." / ".$hgt_nbHGT_tab{"$elt"}. " ";
        my $tmp2 = $tmp . " Trivial";
        if(( $tmp =~ $hgt_number_tab{"$elt"}) ||($tmp2 =~ $hgt_number_tab{"$elt"})){
            if ($mode eq 'mode=multicheck') {
                print OUT "\n=================================================";
                print OUT "\n| After this iteration the criteria values are as
                    follows :";
                print OUT "\n| " . $hgt_pos2[$nbHGT2++] ;
                print OUT "\n=================================================\n";
            }
            $newGroup=0;
        }
    }
    print OUT "\nTotal number of HGTs : $total_hgt ";
    print OUT "(". ($total_hgt-$total_trivial) ." regular + " . $total_trivial .
        " trivial HGTs)" if( $total_trivial > 0);
    $val_retour = $total_hgt;
    open(OUTWEB,">>$outputWeb");

    if($bootstrap eq "yes"){
        print OUTWEB "\nbootHGT = ";
        my $first=0;
        foreach my $elt( @hgt_pos){
           if(($bootstrap eq "yes")&&($hgt_number_tab{"$elt"} !~ "Trivial")){
                if($first==0){
                    printf(OUTWEB "%3.01f",$hgt_compteur_tab{$elt}*100/$nbTrees);
                }
                else{
                    printf(OUTWEB ",%3.01f",$hgt_compteur_tab{$elt}*100/$nbTrees);
                 }
                $first=1;
            }
            elsif($bootstrap eq "yes"){
                if($first==0){
                    printf OUTWEB "0";
                }
                else{
                    printf OUTWEB ",0";
                }
                $first=1;
            }
        }
    }
    close OUTWEB;
}
# Routine pour afficher les résultats de la version consensus
sub print_result_consensus{
    my $nbHGT2=0;
    my $newGroup=0;
    my $cpt=0;
    my @tmp_tab = split(",",$hgt_tab[0]);
    my @out =  sort par_elt @hgt_pos;
    print OUT    "| Program : HGT Detection 3.3 - March, 2011  |\n";
```

```
    print OUT   "| Authors   : Alix Boc, Alpha Boubabcar Diallo and Vladimir
        Makarenkov  |\n";
    print OUT   "|              (Universite du Quebec a Montreal)  |\n";
    print OUT    "| This program computes a unique scenario of horizontal
        gene transfers (HGT) for |\n";
    print OUT   "| the given pair of species and gene phylogenetic trees.  |\n";
    print OUT    "=========================================\n";
    open(OUTCONS,">$outputConsensus") || die("Erreur d'ouverture de outputConsensus") ;
    foreach my $elt( @out){
        printf(OUT "\n(consensus value = %d  of %d = %3.01f%) ",
            $hgt_compteur_tab{"$elt"},$nbTrees,  (($hgt_compteur_tab{"$elt"} * 100 )
             / $nbTrees ));
         print OUT "\n "  . $hgt_description_tab{"$elt"};
         if ( (($hgt_compteur_tab{"$elt"} * 100 ) / $nbTrees )> $cons){
            (my @tab_cons) = ($hgt_description_tab{"$elt"} =~ /\([^)]+\)/g);
            foreach my $espece( @tab_cons){
                $nb = ($espece =~ tr/,//) + 1 ;
                $espece =~ s/\(/ /;
                $espece =~ s/\)/ /;
                $espece =~ s/,/ /g;
                printf OUTCONS "\n". $nb . $espece;
            }
        }
    }
    my $cmd_cons = "hgt_find_branches -inputfile=$inputfile_bkp -path=$path";
    $cmd_cons          = "usagers/" . $cmd_cons;
    execute_hgt("$cmd_cons >> $log_file1");
    close OUTCONS;
    $val_retour = scalar(@out);
    print OUT "\n\n\nTotal number of HGTs : $val_retour ";
    open(OUTWEB,">>$outputWeb");
    printf(OUTWEB "\nbootHGT=");
    my $first=0;
    foreach my $elt( @out){
        if($first==0){
            printf(OUTWEB "%3.01f",$hgt_compteur_tab{$elt}*100/$nbTrees);
         }
         else{
            printf(OUTWEB ",%3.01f",$hgt_compteur_tab{$elt}*100/$nbTrees);
        }
        $first=1;
    }
    close OUTWEB;
}

# Routine pour afficher le titre
sub print_title{
    print STDOUT "=============================================================\n";
    print STDOUT "| HGT-DETECTION V.3.3 (Janvier, 2011) by Alix Boc, Alpha
        Boubacar Diallo and Vladimir Makarenkov |\n";
    print STDOUT "=============================================================\n";
}

# Routine pour afficher la documentation
sub print_minidoc{
    print STDOUT "\nCheck the file $log_file for the computation details";
    print STDOUT "\nCheck the file $outputfile for the program output\n";
}

# Routine pour afficher la description
sub print_description{
    print STDOUT "=====================================================\n";
    print STDOUT "| Program : HGT Detection 3.3 - Janvier, 2011     |\n";
    print STDOUT    "| Authors   : Alix Boc, Alpha Boubacar Diallo and Vladimir
        Makarenkov (Universite du Quebec a Montreal)  |\n";
    print STDOUT "| This program computes a unique scenario of horizontal
        gene transfers (HGT) for                      |\n";
    print STDOUT    "| the given pair of species and gene phylogenetic trees.|\n";
         print STDOUT     "=====================================================\n";
}

# Routine pour afficher l'aide
sub print_help{
    print STDOUT "\nUsage :\nrun_hgt.pl -inputfile=[inputfilename]
```

```
        -outputfile=[outputfilename] -criterion=[rf|ls|bd]";
    print STDOUT "-speciesroot=[midpoint|file]
        -generoot=[midpoint|file|bestbipartition]";
    print STDOUT "-scenario=[unique|multiple] -nbhgt=[maxhgt] -path=[path]
        -bootstrap=[no|yes]";
    print STDOUT "\n\nsee README.txt file for more detail.";
}
```

Tableau C.2 - Code source du script Perl *run_hgt_complets.pl*.

C.3 Le programme *HGT-Detection* (hgt.cpp)

HGT-Detection (hgt.cpp) est le programme de détection de transferts horizontaux de gènes complets. Le programme *HGT-Detection* prend en entrée les paramètres suivants : l'arbre d'espèces et l'arbre de gène, ainsi que différentes options sur le processus de détection de transferts horizontaux de gènes complets telles que le critère d'optimisation, le seuil minimal du bootstrap ou du consensus, etc. Il fournit en sortie : le fichier contenant les transferts détectés et le fichier des statistiques sur les transferts détectés. La version originale de ce programme a été développée par Alix Boc; nous présentons dans cette annexe une version dans laquelle j'ai corrigé des bogues de la version originale liés à la gestion de la mémoire et que j'ai parallélisé par la suite.

Exemple de la ligne de commande :

```
Hgt.exe -path=/home/diallo_a/ -inputfile=input.txt -outputfile=output.txt;
```

Vu la taille du code source, nous ne présenterons que la fonction *main* du programme (voir le Tableau C.3).

```
//============================================================================
//=  HGT-DETECTION v3.3b
//=  Authors : Alix Boc, Alpha Boubacar Diallo and Vladimir Makarenkov
//=  Date    : Janvier 2011
//=
//=  Description : This program detect horizontal gene transfers (HGTs).
//=  As input it takes two   trees : a species tree and a gene tree. The goal is
//=  to transform the species tree into the gene tree by series of SPR operations.
//=  There are 3 criteria : the Robinson and Foulds distance, the least-square
//=  criterion and the bipartition dissimilarity.
//=  We also use the subtree constraint.
//=
//=  input : file with species tree and gene tree in the Newick format.
//=          In the case of simulations, the species tree and all the gene trees
//=          should be in the same file in the Phylip format or as a Newick string
//=  output: a list of HGT and the criteria values for each transfer.
//=
//============================================================================
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <time.h>
#include <signal.h>

#pragma warning(disable:4996)

#include "structures.h"
#include "utils_tree.cpp"
#include "fonctions.cpp"
#define binaireSpecies 0
#define binaireGene    1
```

```
void traiterSignal(int sig){
   printf("\nMESSAGE : SEGMENTATION FAULT #%d DETECTED",sig);
   printf("\nUse valgrind or gdb to fix the problem");
   printf("\n");
   exit(-1);
}
//==========================================================================
//========================================= MAIN ===========================
//==========================================================================
int main(int nargc,char **argv){
   struct InputTree SpeciesTree;     //== initial species tree
   struct InputTree SpeciesTreeCurrent; //== initial species tree
   struct InputTree FirstTree;
   struct InputTree FirstTree2;
   struct InputTree GeneTree;       //== initial gene tree
   struct InputTree SpeciesTreeRed;//== reduced species tree
   struct InputTree GeneTreeRed;    //== reduced gene tree
   struct ReduceTrace aMap;         //== mapping structure between species tree and
                                    //== reduced species tree
   struct InputTree geneTreeSave;
   struct HGT * bestHGTRed = NULL;  //== list of HGTs for the reduced tree
   struct HGT * bestHGT = NULL;     //== list of HGTs for the regular tree
   struct HGT * outHGT = NULL;
   struct HGT * bestHGTmulticheck = NULL;
   int nbHGT_boot;
   int first = 1,k,l;
   int cpt_hgt,i,j,tmp,nbTree=0;
   int bootstrap = 0;
   int multigene = 0;
   int nbHgtFound = 0;
   struct CRITERIA * multicheckTab=NULL;
   struct CRITERIA aCrit;        //== struture of all the criteria
   struct DescTree *DTSpecies,   //== structure of submatrices for the species tree
                   *DTGene;      //== structure of submatrices for the gene tree
   struct Parameters param;
   FILE *in,*out;
   int max_hgt,nbHGT;
   int ktSpecies;
   int trivial = 1;
   int *speciesLeaves = NULL;
   int RFref;
   int imc;
   char *mot = (char*)malloc(100);
   int nomorehgt=0;
   initInputTree(&geneTreeSave);

   //== read parameters
   printf("\nhgt : reading options");
   if(readParameters(&param,argv,nargc)==-1){
      printf("\nhgt : no options specified, see the README file for more details\n");
      exit(-1);
   }
   rand_bootstrap = param.rand_bootstrap;
   signal(SIGSEGV,traiterSignal);

   //== open the input file
   if((in=fopen(param.inputfile,"r"))==NULL){
      printf("\nhgt : The file %s does not exist",param.inputfile);
      exit(-1);
   }

   if(strcmp(param.speciesroot,"file") == 0){
      if(!file_exists(param.speciesRootfileLeaves)){
         printf("\nhgt : The file %s does not exist",param.speciesRootfileLeaves);
         exit(-1);
      }
   }
   if(strcmp(param.generoot,"file") == 0){
      if(!file_exists(param.geneRootfileLeaves)){
         printf("\nhgt : The file %s does not exist",param.geneRootfileLeaves);
         exit(-1);
      }
   }
```

```
   if((in=fopen(param.inputfile,"r"))==NULL){
      printf("\nhgt : Cannot open input file (%s)",param.inputfile);
      exit(-1);
   }
   //== open the bootstrapFile
   if(strcmp(param.bootstrap,"yes") == 0){
      bootstrap = 1;
   }
   if(strcmp(param.multigene,"yes") == 0){
      multigene = 1;
      if(strcmp(param.speciesroot,"file"))
         strcpy(param.speciesroot,"midpoint");
   }
   remove(param.hgtResultFile);

   initInputTree(&FirstTree);
   initInputTree(&SpeciesTreeCurrent);

   FILE * results, *results2;
   if((results = fopen(param.results,"w+"))==NULL){
      printf("\nhgt : Cannot open input file (%s)",param.results);
      exit(0);
   }
   if((results2 = fopen(param.results2,"w+"))==NULL){
      printf("\nhgt : Cannot open input file (%s)",param.results2);
      exit(0);
   }

//=============================================================================
//============================= LECTURE DES ARBRES ============================
//=============================================================================

   printf("\nhgt : reading the input file");
   tmp = readInputFile(in,param.input/*,&SpeciesTree,&GeneTree*/,param.errorFile);

   if(tmp==-1) {
      printf("\nCannot read input data !!\n");
      exit(-1);
   }

   cpt_hgt = 0;
   initInputTree(&SpeciesTree);
   initInputTree(&GeneTree);
   initInputTree(&SpeciesTreeRed);
   initInputTree(&GeneTreeRed);

   //== lecture des matrices ou des chaines Newick en entréee
   if(readInput(SPECIE,param.input,&SpeciesTree) == -1){
      printf("\nError in species tree\n"); exit(-1);
   }

   if(readInput(GENE,param.input,&GeneTree) == -1){
      printf("\nError in gene tree\n"); getchar(); exit(-1);
   }

   TrierMatrices(GeneTree.Input,GeneTree.SpeciesName,SpeciesTree.SpeciesName,
        SpeciesTree.size);
   NJ(SpeciesTree.Input,SpeciesTree.ADD,SpeciesTree.size);
   NJ(GeneTree.Input,GeneTree.ADD,GeneTree.size);

   //== Construction des differentes représentations des arbres (adjacence,arêtes,
   //== longueurs,degré)
   CreateSubStructures(&SpeciesTree,1,binaireSpecies);
   CreateSubStructures(&GeneTree,1,binaireGene);

//=============================================================================
//============================= GESTION DES RACINES ===========================
//=============================================================================
   //== sélection de la racine
   printf("\nhgt : adding the tree roots");
   if(strcmp(param.load,"yes") == 0 ){
      chargerFichier(&SpeciesTree,param.speciesTree,param.speciesRootfile);
      chargerFichier(&GeneTree,param.geneTree,param.geneRootfile);
   }
```

```
else{
   if((strcmp(param.version,"web")==0) && (strcmp(param.printWeb,"yes")==0)){
       saveTree(param.speciesTreeWeb,SpeciesTree,bestHGT,0,cpt_hgt,"",
         param.scenario,NULL);
       saveTree(param.geneTreeWeb,GeneTree,bestHGT,0,cpt_hgt,"",
         param.scenario,NULL);
   }
   if(first == 1){
      int nbBranche,leave;
   }
   if(speciesLeaves == NULL){
      speciesLeaves = (int*) malloc(SpeciesTree.size*sizeof(int));
      speciesLeaves[0] = -1;
   }

   if(SpeciesTree.Root == -1)
      addRoot(&SpeciesTree,NULL,SpeciesBranch,param.speciesroot,
              param.speciesRootfile,param.speciesRootfileLeaves,NULL,
                 param.version);
      if(GeneTree.Root == -1)
         addRoot(&GeneTree,NULL,GeneBranch,param.generoot,param.geneRootfile,
                 param.geneRootfileLeaves,NULL,param.version);

      if(strcmp(param.viewtree,"yes")==0)
         exit(0);
   }
   nbTree++;
   if(SpeciesTree.size > GeneTree.size) max_hgt = 4*GeneTree.size * GeneTree.size;
   else max_hgt = 4*SpeciesTree.size * SpeciesTree.size;

   bestHGTRed = (struct HGT*)malloc(max_hgt*sizeof(struct HGT));
   bestHGT = (struct HGT*)malloc(max_hgt*sizeof(struct HGT));

   for(i=0;i<max_hgt;i++){
      bestHGTRed[i].listSource = NULL;
      bestHGTRed[i].listDestination = NULL;
      bestHGT[i].listSource = NULL;
      bestHGT[i].listDestination = NULL;
   }


   if(first==1 && strcmp(param.scenario,"multiple")!=0 && strcmp(param.mode,
         "multicheck")==0)
   multicheckTab = (struct CRITERIA *)malloc(max_hgt*sizeof(struct CRITERIA));

   if(first==1){
      multicheckTab[0].m = 0;
      imc=0;
   }
   InitCriteria(&aCrit,SpeciesTree.size);
   DTSpecies = (struct DescTree*)malloc((2*SpeciesTree.size-2-SpeciesTree.kt+1)*
                                        sizeof(struct DescTree));

   RechercherBipartition(SpeciesTree.ARETE,SpeciesTree.ADD,SpeciesTree.Root,
       SpeciesTree.Adjacence,DTSpecies,SpeciesTree.size,SpeciesTree.kt);

   if(first ==1){
      FirstTree.ADD=NULL;
      FirstTree.ARETE=NULL;
      copyInputTree(&FirstTree,SpeciesTree,1,1);
      AdjustBranchLength(&FirstTree,GeneTree,binaireSpecies,1);
   }

   InitCriteria(&aCrit,SpeciesTree.size);
   computeCriteria(FirstTree.ADD,GeneTree.ADD,FirstTree.size,&aCrit,
       FirstTree.LONGUEUR,FirstTree.ARETE,GeneTree.LONGUEUR,GeneTree.ARETE);
   AdjustBranchLength(&SpeciesTree,GeneTree,binaireSpecies,1);

   if((bootstrap != 1) || (bootstrap==1 && first==1)){
      fprintf(results,"%d,%lf,%lf\n",aCrit.RF,aCrit.LS,aCrit.BD);
      RFref=aCrit.RF;
   }
```

```
//=====================================================================
//== Ajout de transferts avant le processus de détection
//=====================================================================
//printf("\nhgt : stepbystep=%s",param.stepbystep);
//if(strcmp(param.stepbystep,"yes") == 0){
//== si le fichier prehgtfile existe,
if(file_exists(param.prehgtfile)){
   FILE *prehgt = fopen(param.prehgtfile,"r");
   printf("\nhgt : adding pre-hgt");
   int pos_source,pos_dest,step,valide,newStep=-1;
   int a1,a2,b1,b2;
   int dedans=0,dedans2=0;
   int nbHGTadd=0;
   while(fscanf(prehgt,"%d%d%d%d%d%d",&step,&a1,&a2,&b1,&b2,&valide) != -1){
      dedans=0;
      SpeciesTreeCurrent.ADD=NULL;
      SpeciesTreeCurrent.ARETE=NULL;
      copyInputTree(&SpeciesTreeCurrent,SpeciesTree,1,1);

      DTSpecies = (struct DescTree*)malloc((2*SpeciesTree.size-2-
                  SpeciesTree.kt+1)* sizeof(struct DescTree));
      RechercherBipartition(SpeciesTree.ARETE,SpeciesTree.ADD,SpeciesTree.Root,
          peciesTree.Adjacence,DTSpecies,SpeciesTree.size,SpeciesTree.kt);
      if (valide == 2){
         int tmp=a1; a1=b1; b1=tmp;
         tmp=a2; a2=b2; b2=tmp;
         valide=1;
      }
      pos_source = pos_dest = -1;
      for(i=1;i<=2*SpeciesTree.size-3-SpeciesTree.kt;i++){
         if((SpeciesTree.ARETE[2*i-1] == a1 && SpeciesTree.ARETE[2*i-2] == a2)
         ||(SpeciesTree.ARETE[2*i-2] == a1 && SpeciesTree.ARETE[2*i-1] == a2))
             pos_source = i;
         if((SpeciesTree.ARETE[2*i-1] == b1 && SpeciesTree.ARETE[2*i-2] == b2)
         ||(SpeciesTree.ARETE[2*i-2] == b1 && SpeciesTree.ARETE[2*i-1] == b2))
             pos_dest = i;
      }
      printf("\nhgt : (branches) %d %d",pos_source,pos_dest);
      if(pos_source != -1 && pos_dest != -1){
         if(isAValidHGT(SpeciesTree,pos_source,pos_dest) == 1){
            dedans=1;
            if(step != newStep){
               if(newStep >= 0){
                  imc++;
               }
               else{
                  imc=1;
               }
               newStep = step;
               multicheckTab[0].m ++;
               multicheckTab[imc].nbHgtFound =0;
            }
            printf("\nhgt : (branches) %d %d",pos_source,pos_dest);
            applyHGT(NULL,&SpeciesTree,pos_source,pos_dest);
            AdjustBranchLength(&SpeciesTree,GeneTree,binaireSpecies,1);
            computeCriteria(SpeciesTree.ADD,GeneTree.ADD,SpeciesTree.size,
               &aCrit,SpeciesTree.LONGUEUR,SpeciesTree.ARETE,GeneTree.LONGUEUR
               GeneTree.ARETE);
            printf("\nhgt : %d-%d -> %d-%d",a1,a2,b1,b2);
            printf("\nhgt : CRITERIA
               RF=%d,LS=%lf,BD=%lf\n",aCrit.RF,aCrit.LS,aCrit.BD);
            multicheckTab[imc].nbHgtFound++;
            cpt_hgt ++;
            bestHGT[cpt_hgt].source_A = a1;
            bestHGT[cpt_hgt].source_B = a2;
            bestHGT[cpt_hgt].dest_A = b1;
            bestHGT[cpt_hgt].dest_B = b2;
            bestHGT[cpt_hgt].valide = valide;
            bestHGT[cpt_hgt].crit = aCrit;
            bestHGT[cpt_hgt].source = pos_source;
            bestHGT[cpt_hgt].destination = pos_dest;
            bestHGT[cpt_hgt].sequence = imc;
            if((bestHGT[cpt_hgt].source_A == bestHGT[cpt_hgt].dest_A) ||
               (bestHGT[cpt_hgt].source_A == bestHGT[cpt_hgt].dest_B) ||
```

```
                (bestHGT[cpt_hgt].source_B == bestHGT[cpt_hgt].dest_A) ||
                (bestHGT[cpt_hgt].source_B == bestHGT[cpt_hgt].dest_B)){
                bestHGT[cpt_hgt].trivial = 1;
                bestHGT[cpt_hgt].valide = TRIVIAL;
             }
             if(valide == 3){
                bestHGT[cpt_hgt].valide = valide;
             }
             multicheckTab[imc].LS = aCrit.LS;
             multicheckTab[imc].RF = aCrit.RF;
             multicheckTab[imc].BD = aCrit.BD;
             multicheckTab[imc].rLS = -1;
             multicheckTab[imc].rRF = -1;
             multicheckTab[imc].rBD = -1;
             findListSpecies(&bestHGT[cpt_hgt],DTSpecies,SpeciesTreeCurrent);

             InitCriteria(&aCrit,SpeciesTree.size);
          }
       }
    }
}
SpeciesTreeCurrent.ADD=NULL;
SpeciesTreeCurrent.ARETE=NULL;
copyInputTree(&SpeciesTreeCurrent,SpeciesTree,1,0);
AdjustBranchLength(&SpeciesTreeCurrent,GeneTree,binaireSpecies,1);
AdjustBranchLength(&SpeciesTree,GeneTree,binaireSpecies,1);
DTGene = (struct DescTree*)malloc((2*GeneTree.size-2-GeneTree.kt+1)*
         sizeof(struct DescTree));
RechercherBipartition(GeneTree.ARETE,GeneTree.ADD,GeneTree.Root,
      GeneTree.Adjacence, DTGene,GeneTree.size,GeneTree.kt);
DTSpecies = (struct DescTree*)malloc((2*SpeciesTree.size-2-SpeciesTree.kt+1)*
      sizeof(struct DescTree));
RechercherBipartition(SpeciesTree.ARETE,SpeciesTree.ADD,SpeciesTree.Root,
      SpeciesTree.Adjacence,DTSpecies,SpeciesTree.size,SpeciesTree.kt);
printf("\nhgt : pre-treatment process");
int tousLesCasSontTraitees = FALSE;
int *tab_tous_les_sommets = (int*)malloc((SpeciesTree.size+1) * sizeof(int));
int *tab_sommets_selectionnes = (int*)malloc((SpeciesTree.size+1) * sizeof(int));
int *tab_branches = (int*)malloc( 4*(GeneTree.size+1) * sizeof(int));
int nb_branches;
int temoin_nouveau_cas;
struct HGT aHGT;

for(i=1;i<=SpeciesTree.size;i++) tab_tous_les_sommets[i] = 0;
tab_tous_les_sommets[0] = FALSE;
while(tousLesCasSontTraitees == FALSE){
   ListeSommets_taille_0(GeneTree.Input,tab_tous_les_sommets,GeneTree.size-1);
   tousLesCasSontTraitees = tab_tous_les_sommets[0];
   if(tousLesCasSontTraitees == FALSE){
      tab_sommets_selectionnes[0] = 0;
      temoin_nouveau_cas=0;
      for(i=1;i<GeneTree.size;i++){
         if(tab_tous_les_sommets[i] == 1){
            if(temoin_nouveau_cas == 0){
               temoin_nouveau_cas=1;
            }
            tab_tous_les_sommets[i] = 2;
            tab_sommets_selectionnes[0] = tab_sommets_selectionnes[0] + 1;
            tab_sommets_selectionnes[tab_sommets_selectionnes[0]] = i;
         }
   }
   if(tab_sommets_selectionnes[0] > 0){
      ListesBranchesPourHGT(tab_sommets_selectionnes,GeneTree.ARETE,
          GeneTree.size,DTGene,tab_branches,&nb_branches);

      while(findBestHGT_nombreLimite(DTGene,DTSpecies,tab_branches,nb_branches,
            GeneTree,SpeciesTree,param,&aHGT) > 0){
         applyHGT(SpeciesTree.ADD,&GeneTree,aHGT.source,aHGT.destination);
         AdjustBranchLength(&GeneTree,SpeciesTree,0,1);
         deleteBipartition(DTGene,GeneTree);
         DTGene = (struct DescTree*)malloc((2*GeneTree.size-2-GeneTree.kt+1)*
              sizeof(struct DescTree));
         RechercherBipartition(GeneTree.ARETE,GeneTree.ADD,GeneTree.Root,
              GeneTree.Adjacence,DTGene,GeneTree.size,GeneTree.kt);
```

```
            ListesBranchesPourHGT(tab_sommets_selectionnes,GeneTree.ARETE,
                  GeneTree.size, DTGene,tab_branches,&nb_branches);
            }
         }
      }
   }
   free(tab_tous_les_sommets);
   free(tab_sommets_selectionnes);
   free(tab_branches);
   InitCriteria(&aCrit,SpeciesTree.size);
   computeCriteria(SpeciesTree.ADD,GeneTree.ADD,SpeciesTree.size,&aCrit,
            SpeciesTree.LONGUEUR,SpeciesTree.ARETE,GeneTree.LONGUEUR,GeneTree.ARETE);
   ReduceTree(SpeciesTree,GeneTree,&SpeciesTreeRed,&GeneTreeRed,&aMap,DTSpecies,DTGene,
            binaireSpecies,binaireGene);
   tmp = 0;

   //================================ DÉTECTION DE TRANSFERTS =================
   if(strcmp(param.version,"consol")==0){
      printf("\n================================================");
      printf("\n| CRITERIA VALUES BEFORE THE DETECTION ");
      printf("\n| RF distance  = %2d",aCrit.RF);
      printf("\n| LS criterion = %2.11f",aCrit.LS);
      printf("\n| BD criterion = %2.11f",aCrit.BD);
      printf("\n================================================\n");
   }

   //=========================================================================
   //============ RECHERCHE DES TRANSFERTS : scénario multiple  ==============
   //=========================================================================
   if(strcmp(param.scenario,"multiple")==0){
      cpt_hgt = findAllHGT(SpeciesTreeRed,GeneTreeRed,param,bestHGTRed);
      for(i=1;i<=cpt_hgt;i++){
         expandBestHGT(bestHGTRed[i],&bestHGT[i],aMap,DTSpecies,SpeciesTree);
      }
      sortHGT(bestHGT,cpt_hgt,param);
      printf("\nhgt : scenario multiple");
      if(cpt_hgt > param.nbhgt) cpt_hgt = param.nbhgt;
   }
   else if(strcmp(param.scenario,"unique")==0){
      printf("\nhgt : scenario unique");
      if( strcmp(param.subtree,"yes") == 0 && strcmp(param.mode,"multicheck")==0 ){

         //=====================================================================
         //=RECHERCHE DE TRANSFERTS : scénario unique - PLUSIEURS PAR TOUR ==
         //=====================================================================
         printf("\nhgt : start of detection");
         trivial = (SpeciesTree.kt == 0)?0:1;
         while( findBestHGTtab(SpeciesTreeRed,GeneTreeRed,param,bestHGTRed,
          &nbHgtFound,&trivial,bootstrap) > 0){
            imc++;
            trivial = (SpeciesTree.kt == 0)?0:1;
            printf("\n\n[%d HGT%s]", nbHgtFound,(nbHgtFound > 1)?"s":"");

            if(first==1){
               multicheckTab[0].m ++; //= nombre d'occurences
               multicheckTab[imc].nbHgtFound = nbHgtFound;
            }
            int temoin_zero=-1;
            int cpt_hgt2 = cpt_hgt;
            for(i=0;i<nbHgtFound;i++){
               if(bestHGTRed[i].crit.RF == 0){
                  temoin_zero = i;
               }
            }
            printf("\nHGT-DETECTION : cpt_hgt= %d",cpt_hgt);
            for(i=0;i<nbHgtFound;i++){
               cpt_hgt++;

               expandBestHGT(bestHGTRed[i],&bestHGT[cpt_hgt],aMap,DTSpecies,
                    SpeciesTree);
               bestHGT[cpt_hgt].sequence = imc;
               bestHGT[cpt_hgt].trivial = 0;
               bestHGTRed[i].listSource = NULL;
               bestHGTRed[i].listDestination = NULL;
```

```
      if((temoin_zero != -1)&&(i!=temoin_zero)){
         bestHGT[cpt_hgt].valide = 0;
         printf("\nHGT-DETECTION : un des transfert met RF=0");
         continue;
      }

      if((bestHGT[cpt_hgt].source_A == bestHGT[cpt_hgt].dest_A) ||
         (bestHGT[cpt_hgt].source_A == bestHGT[cpt_hgt].dest_B) ||
         (bestHGT[cpt_hgt].source_B == bestHGT[cpt_hgt].dest_A) ||
         (bestHGT[cpt_hgt].source_B == bestHGT[cpt_hgt].dest_B)){
         bestHGT[cpt_hgt].trivial = 1;
         bestHGT[cpt_hgt].valide = TRIVIAL;
      }

      if((bestHGT[cpt_hgt].source_A == bestHGT[cpt_hgt].dest_A &&
         SpeciesTree.degre[bestHGT[cpt_hgt].source_A] == 3) ||
         (bestHGT[cpt_hgt].source_A == bestHGT[cpt_hgt].dest_B &&
         SpeciesTree.degre[bestHGT[cpt_hgt].source_A] == 3) ||
         (bestHGT[cpt_hgt].source_B == bestHGT[cpt_hgt].dest_A &&
         SpeciesTree.degre[bestHGT[cpt_hgt].source_B] == 3) ||
         (bestHGT[cpt_hgt].source_B == bestHGT[cpt_hgt].dest_B &&
         SpeciesTree.degre[bestHGT[cpt_hgt].source_B] == 3) ){

         bestHGT[cpt_hgt].valide = 0;
         printf("\nhgt : useless=> HGT #%d : [%2d--%2d] -> [%2d--%2d]",cpt_hgt,
             bestHGT[cpt_hgt].source_A,bestHGT[cpt_hgt].source_B,
             bestHGT[cpt_hgt].dest_A,bestHGT[cpt_hgt].dest_B);
         continue;
      }
   }

   initInputTree(&FirstTree2);
   copyInputTree(&FirstTree2,SpeciesTree,0,0);
   cpt_hgt -= nbHgtFound;
   printf("\nHGT-DETECTION : cpt_hgt= %d",cpt_hgt);
   for(i=0;i<nbHgtFound;i++){
      cpt_hgt++;
      if(bestHGT[cpt_hgt].valide > 0){
         SpeciesTree.ADD = NULL;
         SpeciesTree.ARETE = NULL;
         copyInputTree(&SpeciesTree,FirstTree2,0,0);

         applyHGT2(GeneTree.ADD,&SpeciesTree,bestHGT[cpt_hgt].source,
                   bestHGT[cpt_hgt].destination);

         computeCriteria(SpeciesTree.ADD,GeneTree.ADD,SpeciesTree.size,
               &aCrit,SpeciesTree.LONGUEUR,SpeciesTree.ARETE,
               GeneTree.LONGUEUR,GeneTree.ARETE);
         bestHGT[cpt_hgt].crit.LS = aCrit.LS;
         bestHGT[cpt_hgt].crit.RF = aCrit.RF;
         bestHGT[cpt_hgt].crit.BD = aCrit.BD;
         printf("\nRF = %d | LS = %1.2lf | BD =
               %1.2lf",bestHGT[cpt_hgt].crit.RF,
               bestHGT[cpt_hgt].crit.LS,bestHGT[cpt_hgt].crit.BD);
         SpeciesTree.ADD = NULL;
         SpeciesTree.ARETE = NULL;
         copyInputTree(&SpeciesTree,FirstTree2,0,0);
         applyHGT2(GeneTree.ADD,&SpeciesTree,bestHGT[cpt_hgt].destination,
                bestHGT[cpt_hgt].source);

         computeCriteria(SpeciesTree.ADD,GeneTree.ADD,SpeciesTree.size,
               &aCrit, SpeciesTree.LONGUEUR,SpeciesTree.ARETE,
               GeneTree.LONGUEUR,GeneTree.ARETE);
         bestHGT[cpt_hgt].crit.rLS = aCrit.LS;
         bestHGT[cpt_hgt].crit.rRF = aCrit.RF;
         bestHGT[cpt_hgt].crit.rBD = aCrit.BD;
         printf("\nrRF = %d | rLS = %1.2lf | rBD =
              %1.2lf",bestHGT[cpt_hgt].crit.rRF,
               bestHGT[cpt_hgt].crit.rLS,bestHGT[cpt_hgt].crit.rBD);
      }
   }

   SpeciesTree.ADD = NULL;
   SpeciesTree.ARETE = NULL;
```

```
    copyInputTree(&SpeciesTree,FirstTree2,0,0);
    cpt_hgt -= nbHgtFound;
    printf("\nHGT-DETECTION : cpt_hgt= %d",cpt_hgt);
    for(i=0;i<nbHgtFound;i++){
       cpt_hgt++;
       if(bestHGT[cpt_hgt].valide > 0){
          printf("\nhgt : HGT #%d : [%2d--%2d] -> [%2d--%2d]",cpt_hgt,
                 bestHGT[cpt_hgt].source_A,bestHGT[cpt_hgt].source_B,
                 bestHGT[cpt_hgt].dest_A,bestHGT[cpt_hgt].dest_B);
          applyHGT2(GeneTree.ADD,&SpeciesTree,bestHGT[cpt_hgt].source,
                 bestHGT[cpt_hgt].destination);
       }
    }

    computeCriteria(SpeciesTree.ADD,GeneTree.ADD,SpeciesTree.size,&aCrit,
     SpeciesTree.LONGUEUR,SpeciesTree.ARETE,GeneTree.LONGUEUR,GeneTree.ARETE);

    printf("\n\nCriteria values after this step :");
    printf("\nRF = %d | LS = %1.21f | BD = %1.21f\n",aCrit.RF,aCrit.LS,
        aCrit.BD,aCrit.QD);

    if(first==1){
       multicheckTab[imc].LS = aCrit.LS;
       multicheckTab[imc].RF = aCrit.RF;
       multicheckTab[imc].BD = aCrit.BD;
       multicheckTab[imc].QD = aCrit.QD;
    }

    if((cpt_hgt >= param.nbhgt)||(aCrit.RF == 0)) break;

    deleteBipartition(DTSpecies,SpeciesTreeCurrent);
    copyInputTree(&SpeciesTreeCurrent,SpeciesTree,1,1);

    DTSpecies = (struct DescTree*)malloc((2*SpeciesTree.size-2-
        SpeciesTree.kt+1)* sizeof(struct DescTree));
    RechercherBipartition(SpeciesTree.ARETE,SpeciesTree.ADD,SpeciesTree.Root,
        SpeciesTree.Adjacence,DTSpecies,SpeciesTree.size,SpeciesTree.kt);
    FreeMemory_InputTreeReduced(&SpeciesTreeRed,SpeciesTreeRed.size);
    FreeMemory_InputTreeReduced(&GeneTreeRed,GeneTreeRed.size);
    initInputTree(&SpeciesTreeRed);
    initInputTree(&GeneTreeRed);
    free(aMap.map);
    free(aMap.gene);
    free(aMap.species);
ReduceTree(SpeciesTree,GeneTree,&SpeciesTreeRed,&GeneTreeRed,&aMap,DTSpecies,
       DTGene,binaireSpecies,binaireGene);

    if(strcmp(param.stepbystep,"yes") == 0) {
       nomorehgt=1;
       break;
    }
}
free(aMap.map);
free(aMap.gene);
free(aMap.species);
deleteBipartition(DTSpecies,SpeciesTreeCurrent);
int retour=0;
int cpt,nbTours,j;
printf("\nhgt : multicheckTab[3].nbHgtFound = %d",multicheckTab[3].nbHgtFound);
printf("\nhgt : Looking for idling hgt");
retour = DeleteUseLessHGT(cpt_hgt,bestHGT,SpeciesTree,FirstTree);
printf("(%d)",retour);
cpt=1;
if(first==1){
   printf("\nhgt : multicheckTab[0].m=%d : ",multicheckTab[0].m);
   for(i=1;i<=multicheckTab[0].m;i++){
      nbTours = multicheckTab[i].nbHgtFound;
      printf("(%d) ",nbTours);
      for(j=1;j<=nbTours;j++){
         printf("%d ",cpt);
         if(bestHGT[cpt].valide == 0){
            multicheckTab[i].nbHgtFound --;
         }
         cpt++;
```

```
            }
         }
      }
   }
   else{
      //=================================================================
      //== RECHERCHE DES TRANSFERTS : scénario unique - UN PAR TOUR ===
      //=================================================================
      int initial=1;
      while(findBestHGT(initial,SpeciesTreeRed,GeneTreeRed,param,
      &bestHGTRed[cpt_hgt+1]) > 0){
         cpt_hgt++;
         expandBestHGT(bestHGTRed[cpt_hgt],&bestHGT[cpt_hgt],
             aMap,DTSpecies,SpeciesTree);
         bestHGT[cpt_hgt].trivial = 0;
         if((cpt_hgt,bestHGT[cpt_hgt].source_A == bestHGT[cpt_hgt].dest_A) ||
            (cpt_hgt,bestHGT[cpt_hgt].source_A == bestHGT[cpt_hgt].dest_B) ||
            (cpt_hgt,bestHGT[cpt_hgt].source_B == bestHGT[cpt_hgt].dest_A) ||
            (cpt_hgt,bestHGT[cpt_hgt].source_B == bestHGT[cpt_hgt].dest_B)){
            bestHGT[cpt_hgt].trivial = 1;
            bestHGT[cpt_hgt].valide = TRIVIAL;
         }
         if((bestHGT[cpt_hgt].source_A == bestHGT[cpt_hgt].dest_A &&
            SpeciesTree.degre[bestHGT[cpt_hgt].source_A] == 3) ||
            (bestHGT[cpt_hgt].source_A == bestHGT[cpt_hgt].dest_B &&
            SpeciesTree.degre[bestHGT[cpt_hgt].source_A] == 3) ||
            (bestHGT[cpt_hgt].source_B == bestHGT[cpt_hgt].dest_A &&
            SpeciesTree.degre[bestHGT[cpt_hgt].source_B] == 3) ||
            (bestHGT[cpt_hgt].source_B == bestHGT[cpt_hgt].dest_B &&
            SpeciesTree.degre[bestHGT[cpt_hgt].source_B] == 3) ){
            bestHGT[cpt_hgt].valide = 0;
            continue;
         }
         bestHGTRed[i].listSource = NULL;
         bestHGTRed[i].listDestination = NULL;

         applyHGT(GeneTree.ADD,&SpeciesTree,bestHGT[cpt_hgt].source,
                  bestHGT[cpt_hgt].destination);
         AdjustBranchLength(&SpeciesTree,GeneTree,0,1);

         computeCriteria(SpeciesTree.ADD,GeneTree.ADD,SpeciesTree.size,&aCrit,
                 SpeciesTree.LONGUEUR,SpeciesTree.ARETE,GeneTree.LONGUEUR,
                 GeneTree.ARETE);
         loadCriteria(aCrit,&(bestHGT[cpt_hgt]));

         if(strcmp(param.version,"consol")==0){
            printf("\nHGT #%d %d--%d -> %d--%d",cpt_hgt,bestHGT[cpt_hgt].source_A,
                 bestHGT[cpt_hgt].source_B,bestHGT[cpt_hgt].dest_A,
                 bestHGT[cpt_hgt].dest_B);
            printf("\nRF = %d, LS = %lf, BD = %lf \n",aCrit.RF,aCrit.LS,
                 aCrit.BD,aCrit.QD);
         }

         if(bestHGT[cpt_hgt].crit.RF == 0) break;
         if(cpt_hgt >= param.nbhgt) break;

         deleteBipartition(DTSpecies,SpeciesTreeCurrent);
         copyInputTree(&SpeciesTreeCurrent,SpeciesTree,1,1);
         DTSpecies = (struct DescTree*)malloc((2*SpeciesTree.size-2-SpeciesTree.kt+1)*
                  sizeof(struct DescTree));
         RechercherBipartition(SpeciesTree.ARETE,SpeciesTree.ADD,SpeciesTree.Root,
                  SpeciesTree.Adjacence,DTSpecies,SpeciesTree.size,SpeciesTree.kt);
         free(aMap.map);
         free(aMap.gene);
         free(aMap.species);
         FreeMemory_InputTreeReduced(&SpeciesTreeRed,SpeciesTreeRed.size);
         FreeMemory_InputTreeReduced(&GeneTreeRed,GeneTreeRed.size);
         initInputTree(&SpeciesTreeRed);
         initInputTree(&GeneTreeRed);

         ReduceTree(SpeciesTree,GeneTree,&SpeciesTreeRed,&GeneTreeRed,&aMap,
                 DTSpecies,DTGene,binaireSpecies,binaireGene);
      }
      initial=0;
```

```
while(findBestHGT(initial,SpeciesTreeRed,GeneTreeRed,param,
&bestHGTRed[cpt_hgt+1])>0){
   cpt_hgt++;
   expandBestHGT(bestHGTRed[cpt_hgt],&bestHGT[cpt_hgt],aMap,DTSpecies
       SpeciesTree);
   bestHGT[cpt_hgt].trivial = 0;
   if((cpt_hgt,bestHGT[cpt_hgt].source_A == bestHGT[cpt_hgt].dest_A) ||
      (cpt_hgt,bestHGT[cpt_hgt].source_A == bestHGT[cpt_hgt].dest_B) ||
      (cpt_hgt,bestHGT[cpt_hgt].source_B == bestHGT[cpt_hgt].dest_A) ||
      (cpt_hgt,bestHGT[cpt_hgt].source_B == bestHGT[cpt_hgt].dest_B)){
      bestHGT[cpt_hgt].trivial = 1;
      bestHGT[cpt_hgt].valide = TRIVIAL;
   }
   if((bestHGT[cpt_hgt].source_A == bestHGT[cpt_hgt].dest_A &&
      SpeciesTree.degre[bestHGT[cpt_hgt].source_A] == 3) ||
      (bestHGT[cpt_hgt].source_A == bestHGT[cpt_hgt].dest_B &&
      SpeciesTree.degre[bestHGT[cpt_hgt].source_A] == 3) ||
      (bestHGT[cpt_hgt].source_B == bestHGT[cpt_hgt].dest_A &&
      SpeciesTree.degre[bestHGT[cpt_hgt].source_B] == 3) ||
      (bestHGT[cpt_hgt].source_B == bestHGT[cpt_hgt].dest_B &&
      SpeciesTree.degre[bestHGT[cpt_hgt].source_B] == 3) ){

      bestHGT[cpt_hgt].valide = 0;
      if(strcmp(param.version,"consol")==0){
         printf("\nuseless=> HGT #%d : [%2d--%2d] -> [%2d--%2d]",cpt_hgt,
               bestHGT[cpt_hgt].source_A,bestHGT[cpt_hgt].source_B,
               bestHGT[cpt_hgt].dest_A,bestHGT[cpt_hgt].dest_B);
      }
      continue;
   }
   bestHGTRed[i].listSource = NULL;
   bestHGTRed[i].listDestination = NULL;

   applyHGT(GeneTree.ADD,&SpeciesTree,bestHGT[cpt_hgt].source,
       bestHGT[cpt_hgt].destination);
   AdjustBranchLength(&SpeciesTree,GeneTree,0,1);
   computeCriteria(SpeciesTree.ADD,GeneTree.ADD,SpeciesTree.size,&aCrit,
       SpeciesTree.LONGUEUR,SpeciesTree.ARETE,GeneTree.LONGUEUR,GeneTree.ARETE);
   loadCriteria(aCrit,&(bestHGT[cpt_hgt]));
   if(strcmp(param.version,"consol")==0){
      printf("\nHGT #%d %d--%d -> %d--%d",cpt_hgt,bestHGT[cpt_hgt].source_A,
            bestHGT[cpt_hgt].source_B,bestHGT[cpt_hgt].dest_A,
            bestHGT[cpt_hgt].dest_B);
      printf("\nRF = %d, LS = %lf, BD =
            %lf\n",aCrit.RF,aCrit.LS,aCrit.BD,aCrit.QD);
   }
   if(bestHGT[cpt_hgt].crit.RF == 0) {
      int retour;
      do{
         retour = DeleteUseLessHGT(cpt_hgt,bestHGT,SpeciesTree,FirstTree);
      }while(retour > 0);
      break;
   }
   if(cpt_hgt >= param.nbhgt) break;

   deleteBipartition(DTSpecies,SpeciesTreeCurrent);
   copyInputTree(&SpeciesTreeCurrent,SpeciesTree,1,1);
   free(aMap.map);
   free(aMap.gene);
   free(aMap.species);
   DTSpecies = (struct DescTree*)malloc((2*SpeciesTree.size-2-SpeciesTree.kt+1)
            *sizeof(struct DescTree));
   RechercherBipartition(SpeciesTree.ARETE,SpeciesTree.ADD,SpeciesTree.Root,
       SpeciesTree.Adjacence,DTSpecies,SpeciesTree.size,SpeciesTree.kt);
   FreeMemory_InputTreeReduced(&SpeciesTreeRed,SpeciesTreeRed.size);
   FreeMemory_InputTreeReduced(&GeneTreeRed,GeneTreeRed.size);
   initInputTree(&SpeciesTreeRed);
   initInputTree(&GeneTreeRed);
   ReduceTree(SpeciesTree,GeneTree,&SpeciesTreeRed,&GeneTreeRed,&aMap,
       DTSpecies,DTGene,binaireSpecies,binaireGene);
}
free(aMap.map);
free(aMap.gene);
free(aMap.species);
```

```
            deleteBipartition(DTSpecies,SpeciesTreeCurrent);
        }
    }


    //=============================================================================
    //============================ TRAITEMENT DES RÉSULTATS =======================
    //=============================================================================
    printf("\nhgt : formatting the results");
    outHGT = (struct HGT*)malloc(2*param.nbhgt*sizeof(struct HGT));
    nbHGT = formatResult(bestHGT,cpt_hgt,outHGT,FirstTree);
    printHGT(results,results_bouba,param.stepbystep,multicheckTab,param.mode,RFref,
             FirstTree,outHGT,nbHGT,NULL,param.subtree,param.bootmin);
    saveTree(param.outputWeb,FirstTree,outHGT,1,nbHGT,param.subtree,
         param.scenario,NULL);
    remove(param.noMoreHgtfile);
    if(nomorehgt == 0){
       FILE * out = fopen(param.noMoreHgtfile,"w+");
       fclose(out);
    }

    //=============================================================================
    //============================ LIBERATION DE LA MÉMOIRE =======================
    //=============================================================================
    deleteBipartition(DTGene,GeneTree);
    FreeMemory_InputTreeReduced(&SpeciesTreeRed,SpeciesTreeRed.size);
    FreeMemory_InputTreeReduced(&GeneTreeRed,GeneTreeRed.size);
    FreeMemory_InputTree(&SpeciesTreeCurrent,SpeciesTreeCurrent.size);
    FreeMemory_InputTree(&GeneTree,GeneTree.size);
    FreeMemory_InputTree(&SpeciesTree,SpeciesTree.size);
    if(bootstrap !=1)
       FreeMemory_InputTree(&FirstTree,FirstTree.size);
    FreeCriteria(&aCrit,SpeciesTree.size);

    for(i=1;i<=cpt_hgt;i++){
       free(bestHGT[i].listSource);
       free(bestHGT[i].listDestination);
    }
    free(bestHGTRed);
    free(bestHGT);

    fclose(results);

    printf("\nhgt : number of HGT(s) found = %d \nhgt : end of computation,
             check the file results.txt for the program output\n",nbHGT);
    exit(nbHGT);
}
```

Tableau C.3 - Code source de la fonction *main* du programme *HGT-Detection.*

C.4 Version parallèle de la fonction de recherche des transferts : *findBestHGTTab()*

La fonction *findBestHGTTab* recherche un ensemble optimal des transferts horizontaux de gènes *complets*. Cette fonction prend en entrée les paramètres suivants : un arbre d'espèces et un arbre de gène. La fonction fournit en sortie la liste des THGs détectés.

Le code source de la fonction est présenté dans le Tableau C.4.

```
   //=========================================================================
   //= VERSION PARALLÈLE DE LA FONCTION DE RECHERCHE DES TRANSFERTS COMPLETS  =
   //=========================================================================
int findBestHGTtab(struct InputTree SpeciesTree, struct InputTree GeneTree,
struct Parameters param, struct HGT *aHGT, int *nbHgtFound, int *initial) {

    struct InputTree tmpTree;
    int i, j, k, l, first = 1, ret = 0, trouve, tabNbHgtFound[numprocs];
    int size = SpeciesTree.size;
    int ktSpecies;
    struct CRITERIA aCrit, aCritRef, aCritRef2;
    struct DescTree *DTSpecies, *DTGene;
    int encore = 0;
    int sommeNbHgtFound=0;
    initInputTree(&tmpTree);
    (*nbHgtFound) = 0;
    printf("\n== NOUVELLE RECHERCHE == [size=%d] + myid %d \n",
        (2 * size - 3 - SpeciesTree.kt), myid);

    /* Initialisation des critères d'optimisation des arbres */
    InitCriteria(&aCrit, size);
    InitCriteria(&aCritRef, size);
    InitCriteria(&aCritRef2, size);
    computeCriteria(SpeciesTree.ADD, GeneTree.ADD, size, &aCrit, SpeciesTree.LONGUEUR,
        SpeciesTree.ARETE, GeneTree.LONGUEUR, GeneTree.ARETE);
    computeCriteria(SpeciesTree.ADD, GeneTree.ADD, size, &aCritRef, SpeciesTree.LONGUEUR,
        SpeciesTree.ARETE, GeneTree.LONGUEUR, GeneTree.ARETE);
    computeCriteria(SpeciesTree.ADD, GeneTree.ADD, size, &aCritRef2,
         SpeciesTree.LONGUEUR,
        SpeciesTree.ARETE, GeneTree.LONGUEUR, GeneTree.ARETE);
    loadCriteria(aCrit, &aHGT[0]);
    DTGene = (struct DescTree*) malloc((2 * GeneTree.size - 2 - GeneTree.kt + 1)
        * sizeof (struct DescTree));

    /* Recherche des bipartitions */
    RechercherBipartition(GeneTree.ARETE, GeneTree.ADD, GeneTree.Root,
        GeneTree.Adjacence,DTGene, GeneTree.size, GeneTree.kt);
    DTSpecies = (struct DescTree*) malloc((2 * SpeciesTree.size - 2 - SpeciesTree.kt + 1)
        * sizeof (struct DescTree));
    RechercherBipartition(SpeciesTree.ARETE, SpeciesTree.ADD, SpeciesTree.Root,
        SpeciesTree.Adjacence, DTSpecies, SpeciesTree.size, SpeciesTree.kt);
    int flag2 = 1;

    /* Boucle principale de la recherche des transferts horizontaux complets */
    do {
        for (i = 1; i < 2 * size - 3 - SpeciesTree.kt; i++) {

            /* Instruction de parallélisation à travers l'opérateur modulo */
            if (i % numprocs == myid) {
                for (j = i + 1; j < 2 * size - 3 - SpeciesTree.kt; j++) {
                    //== is it a valid hgt ?
                    trouve = 0;

                    /* Vérification des contraintes d'évolution et application du
```

```
        transfert direct */
    if (isAValidHGT(SpeciesTree, i, j) == 1 && i != j) {
        copyInputTree(&tmpTree, SpeciesTree, 0, 0);
        if (strcmp(param.subtree, "yes") == 0)
            if (TestSubTreeConstraint(SpeciesTree, i, j,
            DTSpecies, DTGene) == 0) {
                continue;
                tmpTree.ADD = NULL;
                tmpTree.ARETE = NULL;
            }
        applyHGT(GeneTree.ADD, &tmpTree, i, j);
        if (strcmp(param.criterion, "ls") == 0)
            AdjustBranchLength(&tmpTree, GeneTree, 0, 1);
        computeCriteria(tmpTree.ADD, GeneTree.ADD, size, &aCrit,
            tmpTree.LONGUEUR, tmpTree.ARETE, GeneTree.LONGUEUR,
            GeneTree.ARETE);
        first = 1;
        loadCriteria(aCritRef, &aHGT[(*nbHgtFound)]);
        if (((aCritRef.RF - aCrit.RF) == 1) && (*initial == 1) && (
        (SpeciesTree.ARETE[2 * i - 1] == SpeciesTree.ARETE[2 * j - 1]) ||
        (SpeciesTree.ARETE[2 * i - 1] == SpeciesTree.ARETE[2 * j - 2]) ||
        (SpeciesTree.ARETE[2 * i - 2] == SpeciesTree.ARETE[2 * j - 2]) ||
        (SpeciesTree.ARETE[2 * i - 2] == SpeciesTree.ARETE[2 * j - 2]))){
            UpdateCriterion(&first, param.criterion, aCrit,
                &aHGT[(*nbHgtFound)], i, j);
            aHGT[(*nbHgtFound)].source_A = SpeciesTree.ARETE[2 * i - 1];
            aHGT[(*nbHgtFound)].source_B = SpeciesTree.ARETE[2 * i - 2];
            aHGT[(*nbHgtFound)].dest_A = SpeciesTree.ARETE[2 * j - 1];
            aHGT[(*nbHgtFound)].dest_B = SpeciesTree.ARETE[2 * j - 2];
            findListSpecies(&aHGT[(*nbHgtFound)], DTSpecies,SpeciesTree);
            trouve = 1;
        } else if (*initial == 0) {
            if (TestCriterionAndUpdate(&first, param.criterion, aCrit,
            &aHGT[(*nbHgtFound)], i, j, 0) == 1) {
                aHGT[(*nbHgtFound)].source_A=SpeciesTree.ARETE[2*i -1];
                aHGT[(*nbHgtFound)].source_B=SpeciesTree.ARETE[2*i - 2];
                aHGT[(*nbHgtFound)].dest_A= SpeciesTree.ARETE[2 * j - 1];
                aHGT[(*nbHgtFound)].dest_B=SpeciesTree.ARETE[2 * j - 2];
                findListSpecies(&aHGT[(*nbHgtFound)], DTSpecies,
                    SpeciesTree);
                trouve = 1;
            }
        }
    }
    /* Vérification des contraintes d'évolution et application du
       transfert inverse */
    if (isAValidHGT(SpeciesTree, j, i) == 1 && i != j) {
        copyInputTree(&tmpTree, SpeciesTree, 0, 0);
        if (strcmp(param.subtree, "yes") == 0)
            if (TestSubTreeConstraint(SpeciesTree, j, i, DTSpecies,
            DTGene) == 0) {
                continue;
                tmpTree.ADD = NULL;
                tmpTree.ARETE = NULL;
            }
        applyHGT(GeneTree.ADD, &tmpTree, j, i);
        if (strcmp(param.criterion, "ls") == 0)
            AdjustBranchLength(&tmpTree, GeneTree, 0, 1);
        computeCriteria(tmpTree.ADD, GeneTree.ADD, size, &aCrit,
            tmpTree.LONGUEUR, tmpTree.ARETE, GeneTree.LONGUEUR,
            GeneTree.ARETE);
        int flag = 0;
        first = 1;
        if (trouve == 0)
            loadCriteria(aCritRef, &aHGT[(*nbHgtFound)]);
        else {
            aHGT[(*nbHgtFound)].crit.diff_bd = fabs(aCrit.BD -
                aHGT[(*nbHgtFound)].crit.BD);
            if ((flag2 == 1) && (fabs(aCrit.BD -
            aHGT[(*nbHgtFound)].crit.BD) <= 1)) {
                if ((aHGT[(*nbHgtFound)].crit.RF - aCrit.RF) >= 2) {
                    trouve = 0;
                    flag = 2;
                } else {
```

```
                                flag = 1;
                            }
                        }
                    }
                    if (flag != 1) {
                        if (((aCritRef.RF - aCrit.RF) == 1) && (*initial == 1) && (
                        (SpeciesTree.ARETE[2*i-1]==SpeciesTree.ARETE[2 * j - 1]) ||
                        (SpeciesTree.ARETE[2*i-1]==SpeciesTree.ARETE[2 * j - 2]) ||
                        (SpeciesTree.ARETE[2*i-2]==SpeciesTree.ARETE[2 * j - 2]) ||
                        (SpeciesTree.ARETE[2*i-2]==SpeciesTree.ARETE[2 * j - 2]))) {
                            UpdateCriterion(&first, param.criterion, aCrit,
                                &aHGT[(*nbHgtFound)], j, i);
                            aHGT[(*nbHgtFound)].source_A=SpeciesTree.ARETE[2*j - 1];
                            aHGT[(*nbHgtFound)].source_B = SpeciesTree.ARETE[2*j- 2];
                            aHGT[(*nbHgtFound)].dest_A = SpeciesTree.ARETE[2 * i- 1];
                            aHGT[(*nbHgtFound)].dest_B = SpeciesTree.ARETE[2 *i- 2];
                            findListSpecies(&aHGT[(*nbHgtFound)], DTSpecies,
                                  SpeciesTree);
                            trouve = 1;
                        } else if (*initial == 0) {
                            if (TestCriterionAndUpdate(&first,param.criterion, aCrit,
                            &aHGT[(*nbHgtFound)], j, i, flag) == 1) {
                                aHGT[(*nbHgtFound)].source_A =
                                     SpeciesTree.ARETE[2*j - 1];
                                aHGT[(*nbHgtFound)].source_B =
                                     SpeciesTree.ARETE[2 * j - 2];
                                aHGT[(*nbHgtFound)].dest_A =
                                     SpeciesTree.ARETE[2 * i - 1];
                                aHGT[(*nbHgtFound)].dest_B =
                                     SpeciesTree.ARETE[2 * i - 2];
                                findListSpecies(&aHGT[(*nbHgtFound)], DTSpecies,
                                     SpeciesTree);
                                trouve = 1;
                            }
                        }
                    }
                }
                if (trouve == 1) {
                    (*nbHgtFound)++;
                }
            }
        }
    }
    encore = 0;
    if ((*nbHgtFound == 0) && (flag2 == 1)) {
        flag2 = 0;
        encore = 1;
    }
    if ((*nbHgtFound == 0) && (*initial == 1)) {
        (*initial) = 0;
        encore = 1;
        flag2 = 1;
    }
} while (encore == 1);
/* Synchronisation des résultats */
MPI_Barrier(MPI_COMM_WORLD);*/
/* Envoie des résultats au premier processeur */
if (myid > 0) {
    /* Envoie du nombre de transferts détectés */
    MPI_Send(nbHgtFound, 1, MPI_INT, 0, 30, MPI_COMM_WORLD);
    if ((*nbHgtFound) > 0) {
        /* Envoie des transferts détectés */
        MPI_Send(aHGT, *nbHgtFound, AM_HGT, 0, 30, MPI_COMM_WORLD);
        for (i = 0; i < (*nbHgtFound); i++) {
            MPI_Send(&aHGT[i].listSource[0], 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
            MPI_Send(&aHGT[i].listDestination[0], 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
            MPI_Send(&aHGT[i].1istSource[0], (aHGT[i].listSource[0] + 1), MPI_INT, 0,
                0, MPI_COMM_WORLD);
            MPI_Send(&aHGT[i].listDestination[0], (aHGT[i].1istSource[0] + 1),
                MPI_INT, 0, 0, MPI_COMM_WORLD);
        }
    }
}
/* Traitement des résultats envoyés au premier processeur */
```

```
    else {
        /* Récupération des résultats envoyés au premier processeur */
        for (i = 1; i < numprocs; i++) {
            /* Récupération du nombre de transferts détectés */
            MPI_Recv(&tabNbHgtFound[i], 1, MPI_INT, i, 30, MPI_COMM_WORLD, &status);
            if (tabNbHgtFound[i] > 0){
                /* Récupération des transferts détectés */
                MPI_Recv(&aHGT[(*nbHgtFound)], tabNbHgtFound[i], AM_HGT, i,
                    30, MPI_COMM_WORLD, &status);
                int j;
                for (j = 0; j < tabNbHgtFound[i]; j++) {
                    int source, dest;
                    MPI_Recv(&source, 1, MPI_INT, i, 0, MPI_COMM_WORLD, &status);
                    MPI_Recv(&dest, 1, MPI_INT, i, 0, MPI_COMM_WORLD, &status);
                    aHGT[j + (*nbHgtFound)].listDestination = (int *)
                        malloc((dest + 1) * sizeof (int));
                    aHGT[j + (*nbHgtFound)].listSource = (int *) malloc((source + 1)
                        * sizeof (int));
                    MPI_Recv(&aHGT[j + (*nbHgtFound)].listSource[0], source + 1,
                        MPI_INT, i, 0, MPI_COMM_WORLD, &status);
                    MPI_Recv(&aHGT[j + (*nbHgtFound)].listDestination[0], dest + 1,
                        MPI_INT, i, 0, MPI_COMM_WORLD, &status);
                }
                (*nbHgtFound) += tabNbHgtFound[i];
            }
        }
    }
    deleteBipartition(DTSpecies, SpeciesTree);
    deleteBipartition(DTGene, GeneTree);
    FreeCriteria(&aCrit, size);
    FreeCriteria(&aCritRef, size);
    FreeMemory_InputTree(&tmpTree, tmpTree.size);
    sommeNbHgtFound = (*nbHgtFound);
    MPI_Bcast (&sommeNbHgtFound, 1, MPI_INT, 0, MPI_COMM_WORLD );
    return sommeNbHgtFound;
}
```

Tableau C.4 - Code source de la version parallèle de la fonction *findBestHGTTab()*.

C.5 Script Perl et un programme C++ pour la génération d'arbres phylogénétiques utilisés dans les différentes simulations

Nous présentons ici le script *Perl* (*process.pl*) ainsi que le fichier *C++* (*simhgt_naire.cpp*) développés pour la génération des arbres phylogénétiques de différentes tailles, incluant différents nombres de transferts. Les arbres générés par ce processus ont été utilisés dans les simulations que nous avons réalisées lors de ce projet doctoral (voir les chapitres II, III et IV de la thèse).

C.5.1 Programme C++ : sim_tree.cpp

Ce programme permet de générer des arbres phylogénétiques aléatoires à travers l'utilisation répétitive la fonction suivante : *void tree_generation(double **DA, double **DI, int n, double Sigma)*, voir le Tableau C.5. Cette dernière permet de générer un arbre phylogénétique binaire aléatoire, représenté par sa matrice de distances. Pour plus de détails voir Kuhner et Felsenstein (1994) ou Makarenkov et Legendre (2004).

```
// This C++ function is meant to generate a random tree distance matrix DA
// of size (nxn) and a distance (i.e. dissimilarity) matrix DI obtained
// from DA by adding a random normally distributed noise with mean 0
// and standard deviation Sigma.

void tree_generation(double **DA, double **DI, int n, double Sigma){
   struct TABLEAU { int V; } **NUM, **A;
   int i,j,k,p,a,a1,a2,*L,*L1,n1;
   double *LON,X0,X,U;
   n1=n*(n-1)/2;

   L=(int *)malloc((2*n-2)*sizeof(int));
   L1=(int *)malloc((2*n-2)*sizeof(int));
   LON=(double *)malloc((2*n-2)*sizeof(double));

   NUM=(TABLEAU **)malloc((2*n-2)*sizeof(TABLEAU*));
   A=(TABLEAU **)malloc((n1+1)*sizeof(TABLEAU*));
   for (i=0;i<=n1;i++){
      A[i]=(TABLEAU*)malloc((2*n-2)*sizeof(TABLEAU));
      if (i<=2*n-3) NUM[i]=(TABLEAU*)malloc((n+1)*sizeof(TABLEAU));

      if ((A[i]==NULL)||((i<=2*n-3)&&(NUM[i]==NULL))) {
       printf("\nData matrix is too large\n");
       exit(1);
      }
    }

   /* Generation of a random additive tree topology T*/
   for (j=1;j<=2*n-3;j++){
    for (i=1;i<=n;i++){
       A[i][j].V=0;
       NUM[j][i].V=0;
     }
```

```
 for (i=n+1;i<=n1;i++)
   A[i][j].V=0;
}
A[1][1].V=1; L[1]=1; L1[1]=2; NUM[1][1].V=1; NUM[1][2].V=0;

for (k=2;k<=n-1;k++){
 p=(rand() % (2*k-3))+1;
 for (i=1;i<=(n*(k-2)-(k-1)*(k-2)/2+1);i++)
  A[i][2*k-2].V=A[i][p].V;

 for (i=1;i<=k;i++){
  a=n*(i-1)-i*(i-1)/2+k+1-i;
  if (NUM[p][i].V==0)
   A[a][2*k-2].V=1;
  else
   A[a][p].V=1;
 }

 for (i=1;i<=k;i++){
   a=n*(i-1)-i*(i-1)/2+k+1-i;
   A[a][2*k-1].V=1;
 }
 for (j=1;j<=k;j++){
  if (j==L[p]){
    for (i=1;i<=2*k-3;i++){
     if (i!=p){
      if (L1[p]>L[p])
         a=floor1((n-0.5*L[p])*(L[p]-1)+L1[p]-L[p]);
      else
         a=floor1((n-0.5*L1[p])*(L1[p]-1)+L[p]-L1[p]);
      if (A[a][i].V==1){
       if (NUM[i][L[p]].V==0)
         a=floor1((n-0.5*L[p])*(L[p]-1)+k+1-L[p]);
       else
         a=floor1((n-0.5*L1[p])*(L1[p]-1)+k+1-L1[p]);
       A[a][i].V=1;
      }
     }
    }
  }
  else if (j!=L1[p]){
   a=floor1((n-0.5*j)*(j-1)+k+1-j);
   if (j<L[p])
     a1=floor1((n-0.5*j)*(j-1)+L[p]-j);
   else
     a1=floor1((n-0.5*L[p])*(L[p]-1)+j-L[p]);

   if (j<L1[p])
     a2=floor1((n-0.5*j)*(j-1)+L1[p]-j);
   else
     a2=floor1((n-0.5*L1[p])*(L1[p]-1)+j-L1[p]);
   for (i=1;i<=2*k-3;i++)
   (
     if((i!=p)&&((A[a1][i].V+A[a2][i].V==2)||((NUM[i][j].V+NUM[i][L[p]].V==0)&&
    (A[a2][i].V==1))||((NUM[i][j].V+NUM[i][L1[p]].V==0)&&(A[a1][i].V==1))))
       A[a][i].V=1;
   )
  }
 }
 for (i=1;i<=k;i++)
  NUM[2*k-2][i].V=NUM[p][i].V;
 NUM[2*k-2][k+1].V=1;

 for (i=1;i<=k;i++)
  NUM[2*k-1][i].V=1;

 for (i=1;i<=2*k-3;i++)
 {
  if (((NUM[i][L[p]].V+NUM[i][L1[p]].V)!=0)&&(i!=p))
   NUM[i][k+1].V=1;
 }

 L[2*k-2]=k+1; L1[2*k-2]=L1[p];
 L[2*k-1]=L1[p]; L1[2*k-1]=k+1;
```

```
    L1[p]=k+1;
   }
   U = 0.1;

   for (i=1;i<=2*n-3;i++)
      LON[i] = -1.0/(2*n-3)*log(U);

   i=1;
   while (i<=2*n-3){
       U = 1.0*rand()/RAND_MAX;  LON[i] = 1.0*LON[i]*(1.0+0.8*(-log(U)));
       if (LON[i]>2*epsilon) i++;
   }

   // Computation of a tree distance matrix (tree metric matrix)
   for (i=1;i<=n;i++){
    DA[i][i]=0;
    for (j=i+1;j<=n;j++){
     DA[i][j]=0;
     a=floor1((n-0.5*i)*(i-1)+j-i);
     for (k=1;k<=2*n-3;k++)
      if (A[a][k].V==1) DA[i][j]=DA[i][j]+LON[k];
     DA[j][i]=DA[i][j];
    }
   }

  for (i=1;i<=n;i++){
   DI[i][i]=0.0;
   for (j=i+1;j<=n;j++){
    X=0.0;
    for (k=1;k<=5;k++){
      X0 = 1.0*rand()/RAND_MAX;
      X=X+0.0001*X0;
    }
    X=2*sqrt(0.6)*(X-2.5);
    U=X-0.01*(3*X-X*X*X);
    DI[i][j]=DA[i][j]+Sigma*U;
    if (DI[i][j]<0) DI[i][j]=0.01;
    DI[j][i]=DI[i][j];
   }
  }

  free (L);
  free(L1);
  free(LON);
  for (i=0;i<=n1;i++){
    free(A[i]);
    if (i<=2*n-3) free(NUM[i]);
  }
  free(NUM);
  free(A);
}
```

Tableau C.5 - Code source de la fonction C++ *tree_generation.*

C.5.2 Script Perl : process.pl

Ce script génère les arbres d'espèces et de gènes en utilisant le programme *C++ sim_tree* (exécutable du fichier *sim_tree.cpp*, voir l'Annexe C.5). Ce script ne prend aucun paramètre en entrée. Il fournit en sortie : les fichiers d'arbres d'espèces et de gènes associés à un nombre de transferts indiqués.

Exemple de la ligne de commande du script :

```
perl process.pl;
```

Le code source du script est présenté dans le Tableau C.6.

```
#!/usr/bin/perl
use strict;
use Time::HiRes qw/gettimeofday/;

#=======================================================
#= DÉCLARATION DES VARIABLES DU PROGRAMME
#=======================================================
my $simhgt_dump = "simhgt.exe.stackdump";
my $cmd;          # Ligne de commande à exécuter
my $retour;      # Valeur retournée par les routines

#=======================================================
#= VÉRIFICATION DES ARGUMENTS DE LA LIGNE DE COMMANDE
#=======================================================
if( scalar @ARGV != 0){
    print "\nErreur\nusage : $0";
    exit 0;
}
#=======================================================
#= INITIALISATION DES PARAMÈTRES DE SIMULATION
#=======================================================
my @nb_species     = (20,40,80,160,320);
my @min_transfers  = (5);
my @max_transfers  = (20);
my @nb_transfers   = (5,10,15,20);
my @nb_simulations = (200,200,200,200,200);
my $nb_branches    = -1;

#=======================================================
#= BOUCLE DE GÉNÉRATION DES ARBRES
#=======================================================
for (my $i=0; $i < scalar @nb_species; $i++){
    for (my $j=0;$j < scalar @nb_transfers; $j++){
      if(($nb_transfers[$j] <= $max_transfers[$i]) && ($nb_transfers[$j] >=
      $min_transfers[$i])){
        for(my $k=1;$k<=$nb_simulations[$i];$k++){
          #=====================================================
          #= SIMULATION DES ARBRES
          #=====================================================
          $cmd = "simhgt.exe " . $nb_species[$i] . " " . $nb_transfers[$j] . " 0 J >
          toto";
          print "\n\n#$k : nb species=" . $nb_species[$i] . " , nb transferts=" .
          $nb_transfers[$j];
          $retour = system($cmd);
          if ((-e $simhgt_dump) || $retour != 0){
              $k--;
              system("del -rf $simhgt_dump");
          }
```

```
    else{
            #=========================================================
            #= MODIFICATION DES ARBRES
            #=========================================================
             $nb_branches    = 0;
            if($nb_branches == -1){
              my $range     =  int(($nb_species[$i]-3)-1);
              $nb_branches = int(rand($range-1)) + 1;
            }
            print "\nTransformation des arbres : ($nb_branches)";
            $cmd = "perl add_n_array_nodes.pl " . $nb_species[$i] . " $nb_branches
            input.txt >> input2.txt";
            system($cmd);
          }
        }
      }
    }
}
```

Tableau C.6 - Code source du script Perl *process.pl*.

ANNEXE D

TYPE DE DONNÉES MPI, ROUTINES MPI ET PRORAMMES POUR LA TRANSFORMATION DES STRUCTURES DE DONNÉES C EN STRUCTURES DE DONÉES MPI

Dans l'annexe D, nous présentons tout d'abord les types de données MPI et les routines incluses dans cette librairie. Ensuite, nous montrerons le script *Perl* développé (*genererMPIDatatypes.pl*) pour l'exécution de la transformation des structures de données *C* en structures de données MPI à travers *Automap* (voir l'Annexe B.1.1). Cette annexe présente aussi le programme *Java* développé (contenant les classes *TransformMpi.java* et *automap_and_autoserial.java*) qui permet d'effectuer l'analyse des dépendances des données.

D.1 Types de données MPI et les routines MPI

Les types de données de MPI et leur équivalent du langage *C* sont présentés dans le Tableau D.1.

Types de données de MPI et leur équivalent en *C*	
MPI_CHAR	signed char
MPI_SHORT	signed short int
MPI_INT	signed int
MPI_LONG	signed long int
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED_SHORT	unsigned short int
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_FLOAT	float

MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double
MPI_BYTE	8 binary digits (**pas d'équivalence directe**)
MPI_PACKED	data packed or unpacked with MPI_Pack() / MPI_Unpack

Tableau D.1 - Équivalence entre les types de données MPI et les types de données du langage *C*.

Le Tableau D.2 contient la totalité des routines MPI définies pour l'environnement de programmation *C*.

Routines MPI		
MPI_Abort	MPI_Errhandler_create	MPI_Errhandler_free
MPI_Errhandler_get	MPI_Errhandler_set	MPI_Error_class
MPI_Error_string	MPI_Finalize	MPI_Get_processor_name
MPI_Init	MPI_Initialized	MPI_Wtick
MPI_Wtime	MPI_Start	MPI_Test
MPI_Bsend	MPI_Bsend_init	MPI_Buffer_attach
MPI_Buffer_detach	MPI_Cancel	MPI_Get_count
MPI_Get_elements	MPI_Ibsend	MPI_Iprobe
MPI_Irecv	MPI_Irsend	MPI_Isend
MPI_Issend	MPI_Probe	MPI_Recv
MPI_Recv_init	MPI_Request_free	MPI_Rsend
MPI_Rsend_init	MPI_Send	MPI_Send_init
MPI_Sendrecv	MPI_Sendrecv_replace	MPI_Ssend
MPI_Ssend_init	MPI_Reduce	MPI_Startall
MPI_Scatterv	MPI_Test_cancelled	MPI_Testall
MPI_Testany	MPI_Testsome	MPI_Wait
MPI_Waitall	MPI_Waitany	MPI_Waitsome
MPI_Allgather	MPI_Allgatherv	MPI_Allreduce
MPI_Alltoall	MPI_Alltoallv	MPI_Barrier
MPI_Bcast	MPI_Gather	MPI_Gatherv
MPI_Op_create	MPI_Op_free	MPI_Scatter
MPI_Reduce_scatter	MPI_Scan	MPI_Group_translate_ranks
MPI_Group_union	MPI_Group_difference	MPI_Group_excl
MPI_Group_compare	MPI_Group_union	MPI_Group_rank
MPI_Group_free	MPI_Group_incl	MPI_Group_intersection
MPI_Group_range_excl	MPI_Group_range_incl	MPI_Group_size

MPI_Comm_split	MPI_Comm_test_inter	MPI_Intercomm_create
MPI_Comm_compare	MPI_Comm_create	MPI_Comm_dup
MPI_Comm_free	MPI_Comm_group	MPI_Comm_rank
MPI_Comm_remote_group	MPI_Comm_remote_size	MPI_Comm_size
MPI_Type_commit	MPI_Type_extent	MPI_Type_contiguous
MPI_Intercomm_merge	MPI_Type_free	MPI_Type_hindexed
MPI_Type_count	MPI_Type_vector	MPI_Type_ub
MPI_Type_hvector	MPI_Type_indexed	MPI_Type_lb
MPI_Type_size	MPI_Type_struct	MPI_Cart_shift
MPI_Cart_coords	MPI_Cart_create	MPI_Cart_get
MPI_Cart_map	MPI_Cart_rank	MPI_Pack
MPI_Cart_sub	MPI_Cartdim_get	MPI_Dims_create
MPI_Graph_create	MPI_Graph_get	MPI_Graph_map
MPI_Graph_neighbors	MPI_Graph_neighbors_count	MPI_Graphdims_get
MPI_Topo_test	MPI_Pack_size	MPI_Unpack
MPI_Address	MPI_Attr_delete	MPI_Attr_get
MPI_Attr_put	MPI_DUP_FN	MPI_Keyval_create
MPI_Keyval_free	MPI_NULL_COPY_FN	MPI_NULL_DELETE_FN
MPI_Pcontrol		

Tableau D.2 - Routines MPI pour l'environnement de programmation *C*.

D.2 Script Perl pour l'exécution automatique de Automap

Nous avons développé un script *Perl*, *genererMpiDatatypes.pl*, pour exécuter *Automap* et pour afficher des messages d'erreurs simples, clairs et précis. Notons que la plupart des messages d'erreurs originaux générés par *Automap* n'étaient pas faciles à interpréter. Ce script prend en paramètre le nom du fichier des structures de données à transformer et le chemin d'accès au fichier. Il génère les deux (principaux) fichiers de *Automap* en sortie : *mpitypes.h* et *mpitypes.inc* (voir l'Annexe B.1.1). Notre script permet aussi d'inclure au

fichier *mpitypes.h* l'instruction «#include mpi.h» pour l'utilisation des routines MPI et au fichier *mpitypes.inc* des variables MPI nécessaires à l'initialisation de l'environnement parallèle.

Le code source du script est présenté dans le Tableau D.3.

```
#!/usr/local/bin/perl
#
# Program to do the obvious
#
#=======================================================
#= VÉRIFICATION DES ARGUMENTS DE LA LIGNE DE COMMANDE
#=======================================================
if( scalar @ARGV != 2){
    print "\nErreur, nombre d'arguments incorrecte";
    exit 0;
}

#Déclaration des variables
my $fichier = $ARGV[0];
my $path= $ARGV[1];
my $source=$path.$fichier;
my $outputfile= $path ."logfile.txt";

if(verifier_input($source)){
        #Exécution du programme d'Automap
        open(OUT,">$outputfile") ||  die "Cannot open $inputfiles";
        print "fichier source : $source";
        print OUT `/home/mpiuser/Automap/AutoMap -v -log -noAL $source`;
        close(OUT);

        # Vérification de la validité du fichier
        open(OUT,"$outputfile") || die "Erreur a l'ouverture de $fichier";
        @file_contenu = <OUT>;
        close(OUT);
        my $valide=0;
        my $j=0;
        for (;$j<(scalar (@file_contenu));) {
                if ($file_contenu[$j] =~ m/AutoMap finished without error/) {
                        $valide=1;
                }
                $j++;
        }
        if($valide==0)
        {
                print "\n Erreur lors de l'execution d'automap...\n\n";
                `rm -rf  logbook.txt log.txt mpitypes.h  mpitypes.inc`;
        }
        else{
                print "\nAutoMap s'est termine sans erreur...\n\n";
                # Modification pour fichier mpitypes.h pour inclure "$fihier" et "mpi.h"
                modifier_output();

                # Copies des fichiers d'Automap sur le path et effacement des fichiers du
                #répertoire de scripts
                `cp  mpitypes.h  mpitypes.inc logbook.txt $path`;
                `rm -rf  logbook.txt  log.txt mpitypes.h  mpitypes.inc`;
        }
}


else{
         print "\nErreurs, dans le fichier sources";
}

sub verifier_input($source){
        open(OUT,"$source") || die "Erreur a l'ouverture de $fichier";
        @file_contenu = <OUT>;
        close(OUT);
```

```
        my $valide=0;
        my $j=0;
        for (;$j<(scalar (@file_contenu));) {
                if ($file_contenu[$j] =~ m/AM_Begin /) {
                        if($valide==0){
                                $valide=1;
                        }
                }
                elsif ($file_contenu[$j] =~ m/AM_End /) {
                        if($valide==2){
                                $valide=3;
                        }
                }
                elsif ($file_contenu[$j] =~ m/typedef struct/) {
                        if($valide==1){
                                $valide=2;
                        }
                }

                $j++;
        }
        return ($valide==3)
}

sub modifier_output{
       # Modification pour fichier mpitypes.h pour inclure « #include mpi.h »
        open(OUT,"mpitypes.h") || die "Erreur a l'ouverture de $fichier";
        @file_contenu = <OUT>;
        close(OUT);
        my $j=0;
        my $contenu= "";
        for (;$j<(scalar (@file_contenu));) {

                if ($file_contenu[$j] =~ m/ifndef __MPITYPES_H__/) {
                        $contenu = $contenu . "\n\n#include \"$fichier\" \n#include
                        \"mpi.h\"\n\n";
                }
                $contenu= $contenu . $file_contenu[$j];
                $j++;
        }
        close(OUT);
        open(OUT,">mpitypes.h") || die "Erreur a l'ouverture de $fichier";
        print OUT $contenu;
        close(OUT);
        # Modification pour fichier mpitypes.inc pour inclure les constantes MPI
        open(OUT,">>mpitypes.inc") || die "Erreur a l'ouverture de $fichier";
        print OUT "\n\n//Declaration des variables MPI\nint myid, numprocs;\nint
        namelen;\nchar processor_name[MPI_MAX_PROCESSOR_NAME];\nMPI_Status status;\n\n";
        close(OUT);
}
```

Tableau D.3 - Code source du script Perl *genererMpiDatatypes.pl*.

D.3 Programme Java développé pour la transformation des structures de données C en structures de données MPI

En utilisant le script *genererMpiDatatypes.pl* (voir l'Annexe D.2) pour le cas de *Automap* et les commandes de la bibliothèque *Autoserial* (voir l'Annexe B.1.2), le programme *Java* que nous avons développé permet d'effectuer la traduction automatique des structures de données *C* en structures de données MPI. Il permet, entre autres, dans le cas d'une transformation à travers *Automap*, de valider le fichier des structures de données à transformer. Il vérifie la

présence des mots clés obligatoires qui doivent guider la traduction des structures de données. Si certains mots clés sont absents du fichier, le programme s'assure de les ajouter avant le lancement du processus de traduction des structures de données. Il s'assure aussi que toutes les structures de données sont définies avec l'instruction *typedef struct*. Cependant, le programme ne corrige pas les erreurs syntaxiques ou lexicales qui pourraient être présentes dans le fichier à transformer. Cette étape de validation de fichier n'est pas nécessaire avec *Autoserial*.

Ce programme contient un seul paquet *transform.mpi*. Il est constitué de deux classes *Java* (*TransformMpi.java* et *automap_and_autoserial.java*) présentées dans le Tableau D.4. Nous avons utilisé l'outil *Javadoc* pour la génération des informations présentées ci-dessous.

Sommaire des classes	
automap_and_autoserial	Transform a *.c* or *.h* file into an *Automap* or *Autoserial* compatible file. See: http://math.nist.gov/mcsd/savg/auto/v3.00/automap_use.html and http://home.gna.org/autoserial/
TransformMpi	Program to extract and reformat the source code (*.h* or *.c* files) to work with Automap. See : http://math.nist.gov/mcsd/savg/auto/v3.00/automap_use.html

Tableau D.4 - Classes Java du programme de transformation automatique des structures de données C en structures de données MPI.

La hiérarchie des paquets se présente comme suit :

- Package Hierarchies:
  - **transform.mpi**

La hiérarchie des classes se présente comme suit :

- java.lang.Object
  - transform.mpi.**automap_and_autoserial**
  - transform.mpi.**TransformMpi**

Le code source de la classe *TransformMpi.java* est présenté dans le Tableau D.5.

```
package transform.mpi;

/**
 * Program to extract and reformat some code (.h or .c) to
 * work with AutoMap
 * http://math.nist.gov/mcsd/savg/auto/v3.00/automap_use.html
 * @author Alpha Boubacar Diallo et  Etienne Lord
 * @since 23 Mai 2011
 */
public class TransformMpi {

    /**
     * Normal command line should be:
     * TransformMpi.jar infile outfile
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        System.out.println("Transform C or H code to Automap/Autoserial compatible
files"
        +". - Alpha Boubacar Diallo, Etienne Lord 2011");
        System.out.println("see for AutoMap: "
        +"http://math.nist.gov/mcsd/savg/auto/v3.00/automap_use.html");
        System.out.println("see for Autoserial: http://home.gna.org/autoserial/");
        if (args.length<2) {
            System.out.println("Usage: program inputfile outputfile [-debug]");
        } else {
            //--Set a flag if we run in debug mode
            if (args.length==3&&args[2].equals("-debug"))
                  automap_and_autoserial.debug=false;
            automap_and_autoserial m=new automap_and_autoserial(args[0], args[1]);
        }
    }
}
```

Tableau D.5 - Code source de la classe *TransformMpi.java*.

Le code source du fichier *automap_and_autoserial.java* est présenté dans le Tableau D.6.

```
package transform.mpi;
import java.io.*;
import java.util.*;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

/**
 * Transform a .c or .h file into an Automap or Autoserial compatible file
 * See:
 * http://math.nist.gov/mcsd/savg/auto/v3.00/automap_use.html
 * http://home.gna.org/autoserial/
 * @author Alpha Boubacar Diallo and Etienne Lord
 * @since May 2011
 */
public class automap_and_autoserial {
    private StringBuilder input_string_buffer;              //--Input string to infile
    private StringBuilder output_string_buffer_automap;     //--Output string to outfile
    private StringBuilder output_string_buffer_autoserial;  //--Output string to outfile
    private StringBuilder sample_automap;   //--Autogenerated Sample for the automap uage
    private StringBuilder sample_autoserial;//--Autogenerated Sample for the autoserial
    public static boolean debug = false;
    private HashMap<String, String> define = new HashMap<String, String>(); //--define #

    //--Pattern to find in the infile the right struct
    private Pattern struc_regex;  //--REGEX: Main regex for the struc to find
    private Pattern class_regex;  //--REGEX: Main regex for the class to find
    private Pattern struc_define; //--REGEX: Pattern to find #define
    private Pattern correct_struc;//--REGEX: Correct struct to match for AutoMap before
                                  // writing to output
    private Pattern struc_nbyte;  //--REGEX: Pattern for the line e.g. blue:5;
    private Pattern struc_array;  //--REGEX: Pattern to find int e.g.a[...]
    private Pattern struc_array_inside; //--REGEX: Pattern to have the inside of the
                                        // a[xxx] -> xxx
    private Pattern struc_comment;//--REGEX: Patern for the struct comment find in file
    private Pattern char_type;  //--REGEX: Pattern for the line e.g. char c; or char c=1;
    private Pattern int_type;   //--REGEX: Pattern for the line e.g. int i; or int i=1;
    private Pattern double_type;//--REGEX: Pattern for the line e.g. double d; or double
                                // d=1;
    private Pattern float_type; //--REGEX: Pattern for the line e.g. float f; or float
                                // f=1;
    private Pattern long_type;  //--REGEX: Pattern for the line e.g. long l; or long l=1;
    private Pattern short_type; //--REGEX: Pattern for the line e.g. long l; or long l=1;
    private Pattern valid_type; //--REGEX: Pattern for the line (ex. C++) e.g.  l; or
                                // long l=1;
    private Pattern equal_type; //--REGEX: Pattern for the line (ex. C++) e.g.  l; or
                                // long l=1;
    private Pattern complex_type;//--REGEX: Pattern for the line (ex. C++) e.g. int c, d,
                                 // e; or int c=1, d, e;

    public automap_and_autoserial(String infile, String outfile) {
        //--Initialize the string buffer
        input_string_buffer = new StringBuilder();

        //--Initialize some search pattern
        //--Main pattern to match for structure (multiline)
        //  struct {...} name;
        //  typedef struct {...} name;
        //  struct name {...};
        //  struct name {...} name2;
        //  struct name {...} name1, name2...;
        struc_regex = Pattern.compile("(struct\\s*(\\S*?)\\s*[{](.*?)[}](.*?);)",
        Pattern.DOTALL);
        class_regex = Pattern.compile
        ("(class\\s*(\\S*?)\\s*[{](.*?)[}](.*?);)", Pattern.DOTALL);

        //--Pattern for valid automap struct (multiline)
        // typedef struct {
        // ...
        // } name;
```

```
        correct_struc = Pattern.compile
        ("typedef\\sstruct\\s[{](.*?)[}](.*?);", Pattern.DOTALL);

        //--Pattern to match e.g. void (...); (single line)
        struc_comment = Pattern.compile("(.*?)[(].*?[)].*?;");

        //--Pattern to match e.g. int a[...]; or a[...][...]...; (single line)
        struc_array = Pattern.compile("(.*?)[\\[](.*?)[\\]](.*?;)(.*)");

        //--Pattern to match e.g. [xxx] -> (single line)
        struc_array_inside = Pattern.compile("[\\[](.*?)[\\]]");

        //--Pattern to match e.g. int blue:5; (single line)
        struc_nbyte = Pattern.compile("((.*?)[:]\\d+?;)(.*)");

        //--Pattern to match e.g. #define (single line)
        struc_define = Pattern.compile("#define\\s+(.*?)\\s+(.*)", Pattern.DOTALL);

        char_type = Pattern.compile("char\\s+(.*?);");
        int_type = Pattern.compile("int\\s+(.*?);");
        double_type = Pattern.compile("double\\s+(.*?);");
        float_type = Pattern.compile("float\\s+(.*?);");
        long_type = Pattern.compile("long\\s+(.*?);");
        short_type = Pattern.compile("short\\s+(.*?);");
        valid_type = Pattern.compile("(char|int|double|float|long|short)\\s+(.*?);");
        complex_type = Pattern.compile("(.*?)\\s+(.*?);");
        equal_type = Pattern.compile("(.*)=");

        //complex_type =Pattern.compile("(.*?)\\s+(.*?)\\s+(.*);");
        //--Actual program workflow
        if (readfile(infile)) {
            if (transform()) {
                if (writeoutput(output_string_buffer_automap, outfile)
                && writeoutput(output_string_buffer_autoserial, outfile + ".autoserial")
                && writeoutput(sample_automap, outfile + ".sample")
                && writeoutput(sample_autoserial, outfile + ".autoserial.sample")){
                    System.out.println("Success Automap/Autoserial structure
                    transform.");
                    System.out.println("Automap output:" + outfile);
                    System.out.println("Automap sample output:" + outfile + ".sample");
                    System.out.println("Autoserial output:" + outfile + ".autoserial");
                    System.out.println("Autoserial sample output:" + outfile +
                    ".autoserial.sample");
                    System.exit(0);
                } else {
                    System.out.println("Unable to write outfile(s) : " + outfile);
                    System.exit(-1);
                }
            } else {
                System.out.println("Unable to transform infile : " + infile);
                System.exit(-1);
            }
        } else {
            System.out.println("Unable to read from infile : " + infile);
            System.exit(-1);
        }
    }

    /**
     * Read and do some preprocessing work to the infile
     * @param infile (in file)
     * @return true if success, false otherwise
     */
    private boolean readfile(String infile) {
        try {
            //--Open the file for reading
            BufferedReader br = new BufferedReader(new FileReader(new File(infile)));
            String buffer = "";            //--Buffer for escape \ line
            System.out.println("===Pre-Processing input file (" + infile + "):");
            //--Read the file line by line
            while (br.ready()) {
                // --Process the string to remove some error
                String stri = br.readLine();
```

```
            //--1.1 handle \ case
            if (stri.trim().isEmpty() && !buffer.isEmpty()) {
                stri = buffer;
                buffer = "";
                Matcher m1 = struc_define.matcher(stri);
                if (m1.find()) {
                    String key = m1.group(1);
                    String value = m1.group(2);
                    //--We only keep valid define key for types
                    if (key.indexOf("(") == -1
                            && key.indexOf(")") == -1
                            && key.indexOf(":") == -1) {
                        define.put(key, value);
                    }
                } //--End if m1.find()
            } //--End case 1.1

            //--1. Remove any line comments: // and single \
            int index = stri.indexOf("//");
            int index_single = stri.indexOf("\\");
            //--case Double bracket //
            if (index != -1) {
                stri = stri.substring(0, index) + "/*" + stri.substring(index + 2)
                +"*/";
                if (debug) {
                    System.out.println("Warning (change of comments): " + stri);
                }
            }
            //-- case  escape \
            if (index_single != -1) {
                if (debug) {
                    System.out.println("Warning (change of single comments): " +
                    stri);
                }
                stri = stri.replace("\\", " ");
                buffer += stri + "\n";
            }
            //--Finally, append to the input buffer
            input_string_buffer.append(stri + "\n");
        }
        br.close();
        System.out.println("===Definition found(s):");
        //--Store in memory some #define found for further use in the transformation
        for (String s : define.keySet()) {
            System.out.println(s + ": " + define.get(s));
        }
        System.out.println("===Done PreProcessing");
        return true;
    } catch (Exception e) {
        return false;
    }
} //--End readfile
/**
 * Transform the input file in input_string_buffer and output it to
 * output_string_buffer
 * Note: respect the AutoMap format
 */
private boolean transform() {
    System.out.println("===Transforming to AutoMap compatible and Autoserial"
    +"compatible:");
    //--Initializing a tmp variable
    String tmp = input_string_buffer.toString();

    //--Initialize a buffer for the structure names..
    ArrayList<String> struct_names = new ArrayList<String>();
    //--Initializing the REGEX matcher
    Matcher m_struct_matcher = struc_regex.matcher(tmp);
    Matcher m_class_matcher = class_regex.matcher(tmp);
    //--Create some counter
    int counter = 1;    //--counter when the structure or class name is unknown...
    int counter_nb_structure = 1; //--counter for the number of structures found
    int counter_nb_class = 1;  //--counter for the number of class found
```

```
//--Start the writing to the ouput buffer
output_string_buffer_automap = new StringBuilder();
output_string_buffer_automap.append("/*~ AM_Begin */\n");

output_string_buffer_autoserial = new StringBuilder();
output_string_buffer_autoserial.append("/*\n    Autoserial generated structure by
Transform to MPI website \n");
output_string_buffer_autoserial.append("    Alpha Boubacar Diallo, Etienne Lord"
+" (2012) \n");
output_string_buffer_autoserial.append("    www.trex_cluster.labunix.uqam.ca"
+"\n");
output_string_buffer_autoserial.append("    See: http://home.gna.org/autoserial/"
+"form more informations. \n*/\n\n");

sample_automap = new StringBuilder();
sample_autoserial = new StringBuilder();

////////////////////////////////////////////////////////////////////////
// Create the Samples (start)
////////////////////////////////////////////////////////////////////////

//--This is an example for AutoMap
sample_automap.append("/*\n    Automap sample generated by Transform to MPI
website \n");
sample_automap.append("    Alpha Boubacar Diallo, Etienne Lord (2012) \n");
sample_automap.append("    www.trex_cluster.labunix.uqam.ca \n*/\n\n");
sample_automap.append("/*\n Usage: 1. Copy the generated mpitypes.h, mpitypes.inc
and preprocess_struct.h to your source code directory\n");
sample_automap.append("        2. Save this file as sample_automap.c\n");
sample_automap.append("        3. Compile using: mpicc -g -o a.out
sample_automap.c\n*/\n\n");
sample_automap.append("#include \"mpitypes.inc\"\n\n");
sample_automap.append("int main(int argc, char* argv[]) {\n\n");
sample_automap.append("/* Declare and initialize the new structures */\n\n");

//--This is an example for Autoserial
sample_autoserial.append("/*\n    Autoserial sample generated by Transform to MPI
website \n");
sample_autoserial.append("    Alpha Boubacar Diallo, Etienne Lord (2012) \n");
sample_autoserial.append("    www.trex_cluster.labunix.uqam.ca \n*/\n\n");
sample_autoserial.append("/*\n Usage: 1. Install the autoserial library from"
+"http://home.gna.org/autoserial/ \n");
sample_autoserial.append("        2. Save this file as sample_autoserial.cpp\n");
sample_autoserial.append("        3. Compile using: mpic++ -Iinclude -g -o a.out
sample_autoserial.cpp -Lautoserial -lautoserial\n*/\n\n");
sample_autoserial.append("#include \"mpi.h\"\n");
sample_autoserial.append("#include <autoserial/autoserial_mpi.h>\n#include"
+"<iostream>\n\nusing namespace autoserial;\n\n");

////////////////////////////////////////////////////////////////////////
//
// Structure (AutoMap, AutoSerial)
////////////////////////////////////////////////////////////////////////

//--Iterate over the input_buffer while we find a correct struct definition
while (m_struct_matcher.find()) {
    //--Output to System.out the structure found
    System.out.println("\nFound structure(s): " + (counter_nb_structure) + "\n"
    + m_struct_matcher.group(1) + "\n");
    //--Store in tmp variables the structure attribute
    String struct_name_tmp = m_struct_matcher.group(4); //--Name found after the
    //{ } -- Note: they have priority over the name found before...

     //--Main reformating function for AutoMap
    String struct_content = remove_notsupported(m_struct_matcher.group(3));

    //--Main reformating function for Autoserial
    String
   struct_content_autoserial = reformating_autoserial(m_struct_matcher.group(3));

    //--Main reformating function for AutoSerial
    String struct_name_before = m_struct_matcher.group(2).trim();
    //--Name found before the { }
```

```
//--If we have a structure but the name (after) is empty...
if (struct_name_tmp.isEmpty()) {
    //--1. Name before is empty?
    if (struct_name_before.isEmpty()) {
        //--2. Yes, we put our name...
        struct_name_tmp = "unknown" + counter++;
    } else {
        struct_name_tmp = struct_name_before;
    }
}
struct_name_tmp = struct_name_tmp.trim();
//--Save the names for the samples (see below)
struct_names.add(struct_name_tmp);

//--Autoserial (Creating the file)
output_string_buffer_autoserial.append("class " + struct_name_tmp + " :
public autoserial::ISerializable\n{\n");
output_string_buffer_autoserial.append("\tAS_CLASSDEF(" + struct_name_tmp
+")\n");
output_string_buffer_autoserial.append("\t\tAS_MEMBERS\n");
output_string_buffer_autoserial.append(struct_content_autoserial);
output_string_buffer_autoserial.append("\tAS_CLASSEND;\n");
output_string_buffer_autoserial.append("};\n\n");

//--Autoserial (sample)
sample_autoserial.append("class " + struct_name_tmp + " : public
autoserial::ISerializable\n{\n");
sample_autoserial.append("\tAS_CLASSDEF(" + struct_name_tmp + ")\n");
sample_autoserial.append("\t\tAS_MEMBERS\n");
sample_autoserial.append(struct_content_autoserial);
sample_autoserial.append("\tAS_CLASSEND;\n");
sample_autoserial.append("};\n\n");
//--Ok, clean up is done, we further proceed only if we don't have any
inclosed union...
if (struct_content.indexOf("union") == -1) {
    //--Handle case where a struct has multiple names
    for (String struct_name : struct_name_tmp.split(",")) {
        String st = "typedef struct {" + struct_content + "} " + struct_name
        + "/*~ AM */;\n\n";
        //--Final Test for ungood structure:
        // 1. Inside {} -> Not good
        // 2. Nothing in struct -> Not good
        // 3. Single type which normaly point to a #define -> Not good if not
        // in stored array
        // 4. left behind single /
        //--Replace #define if found -- for the moment, we ignore them and we
        comment the code...
        for (String def : define.keySet()) {
            for (String sti : st.split("\n")) {
                if (sti.trim().startsWith(def)) {
                    if (sti.trim().startsWith(def + ";")) {
                        st = st.replace(def + ";", "/* (replacement of " +
                        def + " using #define not supported)*/");
                    } else {
                        st = st.replace(def, "/* (replacement of " + def + "
                        using #define not supported)*/");
                    }
                }
            }
        }
        //--Misc. corrections and verifications
        Matcher m_correct_struc = correct_struc.matcher(st);
        //--Correct Automap struct&
        boolean b_correct = m_correct_struc.find();
        String s_correct = (b_correct ? m_correct_struc.group(1) : "");
        //--Empty structured?
        boolean empty = true;
        for (String s : s_correct.split("\n")) {
            String strim = s.trim();
            if (!strim.isEmpty() && !strim.startsWith("/*")) {
                empty = false;
            }
        }
```

```
                // Ok, everything is FINE! We write to the output buffer!
                if (!empty && b_correct && s_correct.indexOf("{") == -1 &&
                s_correct.indexOf("}") == -1 && s_correct.indexOf("union") == -1) {
                    output_string_buffer_automap.append(st);
                    // Otherwise, display a problem...
                } else {
                    System.out.println("Unable to create a correct struct for : " +
                    struct_name);
                    //--So we add the structure but commented out...
                    output_string_buffer_automap.append("/* Invalid struct : " +
                    struct_name.replace("\n", "") + " */\n");
                }
            }
        }
        counter_nb_structure++;
    }
    if (counter_nb_structure == 1) {
        System.out.println("Warning: No struct found.");
    }
    System.out.println("Transformation done.");

    ////////////////////////////////////////////////////////////////////////
    // Construction of a C class for Autoserial
    ////////////////////////////////////////////////////////////////////////
    //--Find class
    while (m_class_matcher.find()) {
        System.out.println("\nFound class(s): " + (counter_nb_class) + "\n"
        + m_class_matcher.group(1) + "\n");
        //--Store in tmp variables the structure attribute
        String class_name_tmp = m_class_matcher.group(4); //--Name found after the {
        } -- Note: they have priority over the name found before...
        //--Main reformating function for AutoMap
        //String class_content=remove_notsupported(m_class_matcher.group(3));

        //--Main reformating function for Autoserial
        String class_content_autoserial =
        reformating_autoserial(m_class_matcher.group(3));

        //--Main reformating function for AutoSerial
        String class_name_before = m_class_matcher.group(2).trim();//--Name found
        before the { }


        //--If we have a structure but the name (after) is empty...
        if (class_name_tmp.isEmpty()) {
            //--1. Name before is empty?
            if (class_name_before.isEmpty()) {
                //--2. Yes, we put our name...
                class_name_tmp = "unknown" + counter++;
            } else {
                class_name_tmp = class_name_before;
            }
        }
        class_name_tmp = class_name_tmp.trim();
        counter_nb_class++;
        //System.out.println(class_content_autoserial);

    } //--End find class

    output_string_buffer_automap.append("/*~ AM_End */\n");

    ////////////////////////////////////////////////////////////////////////
    //
    // Create the Samples (end)
    ////////////////////////////////////////////////////////////////////////
    //--Sample automap
    for (int i = 0; i < struct_names.size(); i++) {
        String name = struct_names.get(i);
        sample_automap.append("\t" + name + " struct" + (i + 1) + ";\n");
    }
    sample_automap.append("\n/* MPI variables declaration */\n");
    sample_automap.append("\tint rank, numprocs;\n");
    sample_automap.append("\n/* MPI initialization */\n");
    sample_automap.append("\tMPI_Init(&argc, &argv);\n");
```

```
        sample_automap.append("\tMPI_Comm_size(MPI_COMM_WORLD, &numprocs);\n");
        sample_automap.append("\tMPI_Comm_rank(MPI_COMM_WORLD, &rank);\n");
        sample_automap.append(" \n/* Building of MPI structures */\n");
        sample_automap.append("\tBuild_MPI_Types();\n\n");
        sample_automap.append("/* Your main program loop and functions */\n");
        sample_automap.append("\n\tint msg_id=0;      /* Unique message identifier */\n");
        sample_automap.append("\n\tif (rank == 0)     /* Processor 0 is the Master "
        +"*/\n\t{\n");
        sample_automap.append("\t\t/* Sample MPI_Send from processor 0 to Y */\n\t\tint "
        +"Y;\n\t\tfor (Y=1; Y<numprocs;Y++) {\n");
        for (int i = 0; i < struct_names.size(); i++) {
            sample_automap.append("\t\t\tMPI_Send(&struct" + (i + 1) + ", 1, MPI_INT, "
            +"Y,msg_id,MPI_COMM_WORLD );\n");
        }
        sample_automap.append("\t\t}\t\n\t}\n\telse\n\t{\n");
        sample_automap.append("\t\t/* Sample MPI_Recv from processor 0 to Y
        */\n\t\tMPI_Status status;\n");
        for (int i = 0; i < struct_names.size(); i++) {
            sample_automap.append("\t\tMPI_Recv(&struct" + (i + 1) + ", 1, MPI_INT, "
            +"0,msg_id,MPI_COMM_WORLD,&status);\n");
        }
        sample_automap.append("\t}\n\n");
        sample_automap.append("/* Free the MPI structures */\n\tFree_MPI_Types();\n");
        sample_automap.append("\n/* Finalize MPI */\n\tMPI_Finalize();\n");
        sample_automap.append("\treturn 0;\n}");
        //--Sample autoserial
        sample_autoserial.append("int main(int argc, char* argv[]) {\n");
        sample_autoserial.append("\n/* MPI variables declaration */\n");
        sample_autoserial.append("\tint rank, numprocs;\n");
        sample_autoserial.append("\n/* MPI initialization */\n");
        sample_autoserial.append("\tMPI_Init(&argc, &argv);\n");
        sample_autoserial.append("\tMPI_Comm_size(MPI_COMM_WORLD, &numprocs);\n");
        sample_autoserial.append("\tMPI_Comm_rank(MPI_COMM_WORLD, &rank);\n");
        sample_autoserial.append(" \n/* Declaration of MPI structures */\n");
        for (int i = 0; i < struct_names.size(); i++) {
            String name = struct_names.get(i);
            sample_autoserial.append("\t" + name + " *obj" + (i + 1) + ";\n");
        }
        sample_autoserial.append("\n/* Your main program loop and functions */\n");
        sample_autoserial.append("\tint msg_id=0;      /* Unique message identifier "
        +"*/\n");
        sample_autoserial.append("\n\tif (rank == 0)    /* Processor 0 is the Master "
        +"*/\n\t{\n");
        for (int i = 0; i < struct_names.size(); i++) {
            String name = struct_names.get(i);
            sample_autoserial.append("\t\tobj" + (i + 1) + "= new " + name + "();\n");
        }
        sample_autoserial.append("\t\t/* Sample MPI_Send from processor 0 to Y"
        +"*/\n\t\tfor (int Y=1; Y<numprocs;Y++) {\n");
        for (int i = 0; i < struct_names.size(); i++) {
            sample_autoserial.append("\t\t\tAS_MPI_Send(*obj" + (i + 1) + ", Y, msg_id, "
            +"MPI_COMM_WORLD);    // Send obj" + (i + 1) + " from 0 to Y\n");
        }
        sample_autoserial.append("\t\t}\t\n\t}\n\telse\n\t{\n");
        sample_autoserial.append("\t\t/* Sample MPI_Recv from processor 0 to Y
        */\n\t\tMPI_Status status;\n");
        for (int i = 0; i < struct_names.size(); i++) {
            sample_autoserial.append("\t\tAS_MPI_Recv((ISerializable*&)obj" + (i + 1)
            +",  0, msg_id, MPI_COMM_WORLD, &status);\n");
        }
        sample_autoserial.append("\t}\n\n");
        sample_autoserial.append("/* Free the MPI structures */\n");
        for (int i = 0; i < struct_names.size(); i++) {
            sample_autoserial.append("\tdelete obj" + (i + 1) + ";\n");
        }
        sample_autoserial.append("\n/* Finalize MPI */\n\tMPI_Finalize();\n");
        sample_autoserial.append("\treturn 0;\n}");
        return true;
    }
```

```
/**
 * Function to remove from struct some unwanted attributes (see below)
 * e.g. int a[...]; void, enum....
 *
 * @param content : string containing a the content (inside {...}) of a struct
 * @return a string of the correct structure content
 */
private String remove_notsupported(String content) {
    StringBuilder st = new StringBuilder();
    for (String stri : content.split("\n")) {
        //--2. Handle line like:
        // char    stream[3+3+3];
        Matcher m_struc_array = struc_array.matcher(stri);
        if (m_struc_array.find()) {
            //if (debug) System.out.println("Warning (number inside []): "+stri);
            Matcher m_stru_array_inside = struc_array_inside.matcher(stri);
            boolean flag_removed = false; //--Flag to know if we removed some
            information in the structures...
            while (m_stru_array_inside.find()) {
                String inside = m_stru_array_inside.group(1);
                //--If it is a number... keep it
                try {
                    Integer.valueOf(inside);
                } catch (Exception e) {
                    // Else... replace it
                    if (debug) {
                        System.out.println("Found unknown array size: " + inside);
                    }
                    stri = stri.replace(inside, "");
                    //System.out.println(stri);
                    flag_removed = true;
                }
            }
            if (flag_removed) {
                stri = stri + "/* warning removed inside array size : " +
                m_struc_array.group() + " */" + m_struc_array.group(4);
                if (debug) {
                    System.out.println("New: " + stri);
                }
            }
        }
        //--3. Handle line like
        // UINT16 blue:5;
        Matcher m_struc_nbyte = struc_nbyte.matcher(stri);
        if (m_struc_nbyte.find()) {
            stri = m_struc_nbyte.group(2) + "; /* warning removed number in "
            +"declararion : " + m_struc_nbyte.group(1) + " */\n"
            +m_struc_nbyte.group(3);
            if (debug) {
                System.out.println("Warning (:d in declaration): " + stri);
            }
        }
        //--4. Remove stange line like:
        //  ID              (FSAPI *id_callback) (FLOAT64 number, ID id, PCSTRINGZ
        // string, MODULE_VAR *source_var, PGAUGEHDR gauge);
        if (struc_comment.matcher(stri).find()) {
            if (debug) {
                System.out.println("Warning (comment out structure) : " + stri);
            }
            if (stri.indexOf("/*") != -1) {
                stri = stri.replace("/*", "").replace("*/", "");
            }
            stri = "/*" + stri + "*/";
        }
        //--5. Removing ENUM from struc:
        if (stri.contains("ENUM") || stri.contains("enum")) {
            if (debug) {
                System.out.println("Warning (comment out ENUM in structure) : "
                +stri);
            }
            if (stri.indexOf("/*") != -1) {
                stri = stri.replace("/*", "").replace("*/", "");
            }
```

```
            stri = "/*" + stri + " (ENUM not supported.) */";
        }
        //--5. Removing struct from struc:
        if (stri.trim().startsWith("struct")) {
            if (debug) {
                System.out.println("Warning (comment out struct found embedded in "
                +"struct) : " + stri);
            }
            if (stri.indexOf("/*") != -1) {
                stri = stri.replace("/*", "").replace("*/", "");
            }
            stri = "/*" + stri + " (embedded struct not supported.) */";
        }
        //--6. Removing void from struc:
        if (stri.trim().startsWith("void")) {
            if (debug) {
                System.out.println("Warning (comment out void in structure) : "
                +stri);
            }
            if (stri.indexOf("/*") != -1) {
                stri = stri.replace("/*", "").replace("*/", "");
            }
            stri = "/*" + stri + " (embedded void not supported.) */";
        }

        //--6. Removing #ifndef from struc:
        if (stri.trim().startsWith("#")) {
            if (debug) {
                System.out.println("Warning (comment out preprocessing (#) in
                structure) : " + stri);
            }
            if (stri.indexOf("/*") != -1) {
                stri = stri.replace("/*", "").replace("*/", "");
            }
            stri = "/*" + stri + " (embedded preprocessing not supported.) */";
        }
        st.append(stri + "\n");
    }
    return st.toString();
}

/**
 * Function to remove from struct some unwanted attributes (see below)
 * e.g. int a[...]; void, enum....
 *
 * @param content : string containing a the content (inside {...}) of a struct
 * @return a string of the correct structure content
 */
private String reformating_autoserial(String content) {
    StringBuilder st = new StringBuilder();
    int ptr_number = 0; //--Counter for the ptr_size if AS_DYNARRAY in structure
    for (String str : content.split("\n")) {
        //--1. Handle line like:
        // char    i,j,k;
        //--By first spliting the line into 3 char i; char j; char k;
        String[] substri = str.trim().split(",");
        //--Found possible multiple declaration on one line
        if (substri.length > 1) {
            if (struc_comment.matcher(str).find() || str.contains("ENUM") ||
            str.contains("enum") || str.trim().startsWith("//")) {
            //--Do nothing - Really just a normal line... -> create a one line buffer
                substri = new String[0];
                substri[0] = str;
            } else {
                //--Ok, the real deal, split the definition
                //1. What we have?
                //--Add ; to the end of substri[0]
                substri[0] = substri[0] + ";";
                Matcher m_type = complex_type.matcher(substri[0]);
                if (m_type.find()) {
                    String ntype = m_type.group(1);
                    for (int i = 1; i < substri.length; i++) {
                        substri[i] = ntype + " " + substri[i].trim();
```

```
                if (!substri[i].endsWith(";")) {
                    substri[i] = substri[i] + ";";
                }
            }
        }
    }
}
//--Find the type and name
String type = "";
String data = "";     //--or name
String len = "";      //--In case of array
for (String stri : substri) {
    //System.out.println("AutoSerial:"+stri);
    Matcher m_valid = valid_type.matcher(stri);
    if (m_valid.find()) {
        type = m_valid.group(1);
        data = m_valid.group(2);
    } else {
        Matcher m_complex = complex_type.matcher(stri);
        if (m_complex.find()) {
            type = m_complex.group(1);
            data = m_complex.group(2);
        }
    }
    //--Final clean up
    //--Remove: =
    if (data.indexOf("=") > -1) {
        data = data.substring(0, data.indexOf("="));
    }
    //--Break if data or type is null
    if (data.isEmpty() || type.isEmpty()) {
        //--Empty data
    } else {
        Matcher m_struc_array = struc_array.matcher(stri);
        //--CASE 1. Array?
        if (m_struc_array.find()) {
            //--Determine the size
            Matcher m_stru_array_inside = struc_array_inside.matcher(stri);
            int length = 1;
            boolean flag_removed = false; //--Flag to know if we removed some
            information in the structures...
            while (m_stru_array_inside.find()) {
                String inside = m_stru_array_inside.group(1);
                //--If it is a number... keep it
                try {
                    int i = Integer.valueOf(inside);
                    length *= i;
                } catch (Exception e) {
                    // Else... replace it
                    flag_removed = true;
                }
            } //--End while
            if (!flag_removed) {
                data = data.substring(0, data.indexOf('[')); //--Keep the
                name til [ bracket
                stri = "\t\tAS_ARRAY(" + type + ", " + data + ", " + length +
                ")";
            } else {
                //--Ok, but the size will be off...
                data = data.substring(0, data.indexOf('[')); //--Keep the
                name til [ bracket
                stri = "\t\tAS_ARRAY(" + type + ", " + data + ", " + length +
                ") /* Warning, unable to find true array size for " + stri +
                ", correct it manually. */";
            }
        } //--CASE 2. No array
        else {
            //--Pointer?
            if (data.startsWith("*")) {
                String new_size = "new_ptr_size" + ptr_number;
                stri = "\t\tAS_ITEM(int, " + new_size + ")\n\t\tAS_DYNARRAY("
                + type + ", " + data.substring(1) + ", " + new_size + ")
```

```
                    /* Note: Autoserial need to know the size of the dynamic
                array. A new variable was introduce above (" + new_size + ") */";
                    ptr_number++;
                } else {
                    stri = "\t\tAS_ITEM(" + type + ", " + data + ")";
                }
            }
            if (m_struc_array.find()) {
                boolean flag_removed = false; //--Flag to know if we removed some
                information in the structures...
                Matcher m_stru_array_inside = struc_array_inside.matcher(stri);
                while (m_stru_array_inside.find()) {
                    String inside = m_stru_array_inside.group(1);
                    //--If it is a number... keep it
                    try {
                        Integer.valueOf(inside);
                    } catch (Exception e) {
                        // Else... replace it
                        stri = stri.replace(inside, "");
                        //System.out.println(stri);
                        flag_removed = true;
                    }
                }
                if (flag_removed) {
                    stri = stri + "/* warning removed inside array size : " +
                    m_struc_array.group() + " */\n" + m_struc_array.group(4);
                }
            }
            //--3. Handle line like
            // UINT16 blue:5;
            Matcher m_struc_nbyte = struc_nbyte.matcher(stri);
            if (m_struc_nbyte.find()) {
                stri = m_struc_nbyte.group(2) + "; /* warning removed number in
                declaration : " + m_struc_nbyte.group(1) + " */\n" +
                m_struc_nbyte.group(3);
            }

            //--4. Removing ENUM from struc:
            if (stri.contains("ENUM") || stri.contains("enum")) {
                if (stri.indexOf("/*") != -1) {
                    stri = stri.replace("/*", "").replace("*/", "");
                }
                stri = "/*" + stri + " (ENUM not supported.) */";
            }

            //--5. Removing struct from struc:
            if (stri.trim().startsWith("struct")) {
                if (stri.indexOf("/*") != -1) {
                    stri = stri.replace("/*", "").replace("*/", "");
                }
                stri = "/*" + stri + " (embedded struct not supported.) */";
            }

            //--6. Removing void from struc:
            if (stri.trim().startsWith("void")) {
                if (stri.indexOf("/*") != -1) {
                    stri = stri.replace("/*", "").replace("*/", "");
                }
                stri = "/*" + stri + " (embedded void not supported.) */";
            }

            //--6. Removing #ifndef from struc:
            if (stri.trim().startsWith("#")) {
                if (stri.indexOf("/*") != -1) {
                    stri = stri.replace("/*", "").replace("*/", "");
                }
                stri = "/*" + stri + " (embedded preprocessing not supported.)
                */";
            }
            st.append(stri + "\n");
        } //--End no data
    } //--End for stri
} //--End for str
```

```
        // (debug) System.out.println(st.toString());
        return st.toString();
    }

    /**
     * Write output_string_buffer to outfile
     * @param outfile (the filename to write to)
     * @return true if success, false otherwise.
     */
    private boolean writeoutput(StringBuilder output, String outfile) {
        try {
            PrintWriter pw = new PrintWriter(new FileWriter(new File(outfile)));
            pw.print(output.toString());
            pw.flush();
            pw.close();
            return true;
        } catch (Exception e) {
            return false;
        }
    }
```

Tableau D.6 - Code source de la classe *automap_and_autoserial.java*.

ANNEXE E

PRORAMME POUR LA DÉTECTION ET L'ANALYSE DES BOUCLES *FOR* DU PROGRAMME À PARALLÉLISER

L'annexe E présente le code source développé en *Java* pour compléter les paquets générés par *SableCC* (voir l'Annexe B.3) afin de réaliser l'analyse des dépendances des données au niveau des boucles à paralléliser. Le programme est composé des treize paquets présentés dans le Tableau E.1. Nous avons utilisé l'outil *Javadoc* pour la génération des informations présentées ci-dessous.

Paquets	
bouclesablecc	Paquet du programme *Java* que nous avons développé
simpleCAdder.analysis	Paquet *Analysis* produit par la grammaire de compilation
simpleCAdder.lexer	Paquet *Lexer* produit par la grammaire de compilation
simpleCAdder.node	Paquet *Node* produit par la grammaire de compilation
simpleCAdder.parser	Paquet *Parser* produit par la grammaire de compilation
simpleCInit.analysis	Paquet *Analysis* produit par la grammaire de pré-compilation (initialisation)
simpleCInit.lexer	Paquet *Lexer* produit par la grammaire de pré-compilation (initialisation)
simpleCInit.node	Paquet *Node* produit par la grammaire de pré-compilation (initialisation)
simpleCInit.parser	Paquet *Parser* produit par la grammaire de pré-compilation (initialisation)
simpleCPreprocessing.analysis	Paquet *Analysis* produit par la grammaire de pré-compilation

simpleCPreprocessing.lexer	Paquet *Lexer* produit par la grammaire de pré-compilation
simpleCPreprocessing.node	Paquet *Node* produit par la grammaire de pré-compilation
simpleCPreprocessing.parser	Paquet *Parser* produit par la grammaire de pré-compilation

Tableau E.1 - Les paquets utilisés par le programme de détection et d'analyse des dépendances des données.

Ce programme permet l'automatisation de l'analyse des dépendances des données. Tout d'abord, il inclut tous les paquets produits à travers nos grammaires *SableCC*. À partir des fonctions contenues dans les paquets produits par *SableCC*, nous récupérons les informations concernant toutes les boucles du programme avant de procéder aux détections des dépendances des données à travers la bibliothèque PLATO ou le compilateur *Cetus*.

Le paquet *bouclesablecc* est constitué des treize classes présentées dans le Tableau E.2 (ce tableau a été généré par *Javadoc*).

Sommaire des classes	
BlankRemover	Classe permettant d'enlever les espaces (whitespaces) dans le code source à analyser.
ExecutorCetus	Classe générant le code source permettant la détection des dépendances par l'utilitaire Cetus et exécutant le test des dépendances spécifiées en entrée.
ExecutorPlato	Classe générant le code source permettant la détection des dépendances par l'utilitaire PLATO et exécutant le test des dépendances spécifiées en entrée.
GlobalDataStore	Classe implémentant une structure contenant les informations sur les différentes boucle *for* détectées pour analyse des dépendances.
InterpreterCAdder	Classe principale de la détection des boucles 'for' basée sur le 'parser' généré par SableCC.
InterpreterCInit	Classe servant à faire une deuxième uniformisation du code source en entrée.
InterpreterCPreprocessing	Classe permettant d'enlever les parties du code source non nécessaires pour l'analyse, tels que l'instruction '#pragma' et les commentaires.
main	Classe principale du paquet *bouclesablecc* qui détecte les boucles 'for' dans un code source et exécute les différentes applications permettant la détection des dépendances.
Makefile	Classe permettant la génération des 'Makefile' utilisées lors de la

	détection des dépendances par PLATO.
StructureBoucleCetus	Classe implémentant une structure contenant les informations sur les boucles *for* traitées lors de l'analyse par l'application le compilateur Cetus.
StructureBouclePlato	Classe implémentant une structure contenant les informations sur les boucles *for* traitées lors de l'analyse par l'application de la librairie PLATO.
VariableData	Classe servant à l'identification des différents composants des structures 'struct'.
WebPage	Classe de génération des pages Web affichées sur le site *transform to MPI*.

Tableau E.2 - Les treize classes qui constituent le paquet *bouclesablecc*.

Le code source de la classe *BlankRemover.java* est présenté dans le Tableau E.3.

```
/* La classe BlankRemover.java */

package bouclesablecc;

/**
 * Classe permettant d'enlever des espaces (whitespaces) dans le code source
 * à analyser.
 * @author Alpha Boubacar Diallo
 * @since February 2011
 */
public class BlankRemover {
    /**
     * Constructeur
     */
    private BlankRemover () {}

    /**
     * Enlève les espaces à l'avant de la chaîne de caractères.
     * @param source
     * @return la nouvelle chaîne de caractères.
     */
    public static String ltrim(String source) {
        return source.replaceAll("^\\s+", "");
    }

     /**
     * Enlève les espaces à l'arrière de la chaîne de caractères.
     * @param source
     * @return la nouvelle chaîne de caractères.
     */
    private static String rtrim(String source) {
        return source.replaceAll("\\s+$", "");
    }

     /**
     * Enlève les espaces multiples entre les différents 'mots' de la
      * chaîne de caractères.
     * @param source
     * @return la nouvelle chaîne de caractères.
     */
    private static String itrim(String source) {
        return source.replaceAll("\\b\\s{2,}\\b", " ");
    }

   /**
     * Enlève les espaces de la chaîne de caractères.
     * @param source
     * @return la nouvelle chaîne de caractères.
     */
    public static String trim(String source) {
        return itrim(ltrim(rtrim(source)));
    }

     /**
     * Enlève les espaces à l'avant et l'arrière de la chaîne de caractères.
     * @param source
     * @return la nouvelle chaîne de caractères.
     */
    public static String lrtrim(String source){
        return ltrim(rtrim(source));
    }
}
```

Tableau E.3 - Le code source de la classe *BlankRemover.java*.

Le code source de la classe *ExecutorCetus.java* est présenté dans le Tableau E.4.

```
/* La classe ExecutorCetus.java */

package bouclesablecc;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
/**
 * Classe générant le code source permettant la détection des dépendances par
 * l'utilitaire Cetus et exécutant le test des dépendances spécifié en entrée.
 * @author Alpha Boubacar Diallo et Etienne Lord
 * @since July 2011
 */
public class ExecutorCetus {
    /**
     * Variable de sortie de l'application Cetus.
     */
    public int exitVal = -99;
    /**
     * Fichier à exécuter (e.g. command.sh).
     */
    private String filename = "";
    /**
     * Variable servant à indiquer le mode déboguage de la classe.
     */
    public static boolean debug = true; // Debug and programming variables
    /**
     * Regex permettant de connaître l'emplacement des erreurs de 'parsing'
     * rencontrées par Cetus.
     * E.g. match : cetus: exception: line 19:2
     */
    Pattern cetus_exception;          //--Match : cetus: exception: line 19:2
    /**
     * Chemin vers le répertoire de sortie de l'application Cetus.
     */
    String outputdir = "";
    /**
     * Chemin vers le fichier d'entrée à traiter par Cetus.
     */
    String inputfile = "";
    /**
     * Variable indiquant le type de test de dépendance à exécuter par Cetus.
     * If true, we will performed the Range test, otherwise the Banerjee-Wolfe test
     */
    public boolean range_test = false;
    /**
     * Constructeur de la classe
     * @param filename the command_filename (.sh) (see createCommandFile)
     * @param outputdir the outputfile for cetus (e.g. /usager/xxx/)
     * @param inputfile the file to process...
     */
    public ExecutorCetus(String filename, String outputdir, String inputfile) {
        this.filename = filename;
        this.outputdir = outputdir;
        this.inputfile = inputfile;
        cetus_exception = Pattern.compile("cetus:.exception:.line ([0-9]*):([0-9]*)");
    }
    /**
     * Fonction créant le fichier (bash) à exécuter pour la détection des dépendances.
     * @return true si aucune erreur.
     */
    public boolean createCommandFile() {
        try {
            BufferedWriter command_file = new BufferedWriter(new FileWriter(new
            File(filename)));
```

```
            String command = "";
            if (range_test) {
                command = "java -classpath \".:usagers/cetus.jar:usagers/antlr.jar\""
                +"cetus.exec.Driver -verbosity=2 -ddt=2 -parallelize-loops=4 -induction"
                +"-outdir=" + outputdir + "cetus_output " + inputfile + "\n";
            } else {
                command = "java -classpath \".:usagers/cetus.jar:usagers/antlr.jar\""
                +"cetus.exec.Driver -verbosity=2-ddt=1 -parallelize-loops=4 "
                +"-induction -outdir=" + outputdir + "cetus_output " + inputfile + "\n";
            }
            command_file.append(command);
            command_file.flush();
            command_file.close();
        } catch (Exception e) {
            return false;
        }
        return true;
    }
    /**
     * Fonction réalisant le test de dépendance en exécutant l'application Cetus.
     * Note: the filename must be specified in the constructor
     * @return the output (System.out/.err) of the execution
     */
    public StringBuilder run() {
        System.out.println("\n**************************************************");
        System.out.println("**       Computing dependency using Cetus 1.3...    **");
        System.out.println("**************************************************\n");
        StringBuilder st = new StringBuilder();
        try {
            exitVal = -99;
            //--Chmod (just to ensure that the file is executable
            Runtime r = Runtime.getRuntime();
            r.exec("chmod +x " + filename);
            ///--Execute the .sh script
            String[] com = new String[5];
            for (int i = 0; i < com.length; i++) {
                com[i] = "";
            }
            //Execute the Makefile_test.command.sh
            com[0] = filename;
            Process p = r.exec(com);
            InputStream stdoutput = p.getInputStream();
            InputStream stderr = p.getErrorStream();
            InputStreamReader isr = new InputStreamReader(stdoutput);
            InputStreamReader isr2 = new InputStreamReader(stderr);
            BufferedReader br = new BufferedReader(isr);
            BufferedReader br2 = new BufferedReader(isr2);
            String line = null;
            //--System.out
            while ((line = br.readLine()) != null) {
                st.append(line + "\n");
                if (debug) {
                    System.out.println(line);
                }
            }
            exitVal = p.waitFor();
            //--System.err
            if (exitVal != 0) {
                System.out.println("Cetus error " + exitVal);
                while ((line = br2.readLine()) != null) {
                    Matcher ce = cetus_exception.matcher(line);
                    if (ce.find()) {
                        int li = Integer.valueOf(ce.group(1));
                        int row = Integer.valueOf(ce.group(2));
                        System.out.println(li + ":" + row);
                        BufferedReader bre = new BufferedReader(new FileReader(new
                        File(inputfile)));
                        int count_line = 1;
                        String stri = "";
                        while (bre.ready() && count_line < li) {
                            stri = bre.readLine();
                            System.out.println(count_line + ":" + stri);
                            count_line++;
                        }
```

```
                    bre.close();
                    System.out.println("Line (" + li + "):" + stri);
                    String unknown_word = stri.substring(row - 1, stri.indexOf(" ",
                    row));
                    System.out.println("Keyword causing error: " + unknown_word);
                    System.exit(101); //--Error cetus
                }
                st.append(line + "\n");
                if (debug) {
                    System.out.println(line);
                }
            }
        }
    } catch (Exception ex) {
        System.out.println(" error unable to execute the test... Error message
        follow:");
        exitVal = -1;
    }
    return st;
} // End run

////////////////////////////////////////////////////////////////////////////
// HELPER FUNCTIONS
/**
 * Fonction supprimant un fichier.
 * @param filename (to delete)
 * @return true if success
 */
public boolean deleteFile(String filename) {
    try {
        File outtree = new File(filename);
        if (outtree.exists()) {
            outtree.delete();
        }
    } catch (Exception e) {
        System.out.println("Unable to delete file " + filename);
        return false;
    }
    return true;
}

/**
 * Fonction vérifiant la présence d'un fichier.
 * @param filename
 * @return true if file Exists
 */
public static boolean FileExists(String filename) {
    File f = new File(filename);
    if (f == null || f.isDirectory()) {
        return false;
    }
    return f.exists();
}
}
```

Tableau E.4 : Le code source de la classe *ExecutorCetus.java*.

Le code source de la classe *ExecutorPlato.java* est présenté dans le Tableau E.5.

```
/* La classe ExecutorPlato.java */

package bouclesablecc;

import java.io.BufferedReader;
import java.io.File;
import java.io.InputStream;
import java.io.InputStreamReader;

/**
 * Classe générant le code source permettant la détection des dépendances par
 * l'utilitaire PLATO et exécutant le test des dépendances spécifié en entrée.
 * @author Alpha Boubacar Diallo et Etienne Lord
 * @since July 2011
 */
public class ExecutorPlato {

    ////////////////////////////////////////////////////////////////////////////
    /// Variales
    /**
     * Variable de sortie de l'application PLATO.
     */
    public int exitVal = -99;  //--Program execution exit value
    /**
     * Fichier à exécuter (e.g. command.sh).
     */
    private String filename = "";//--Filename to execute (run()) e.g. makefile script.sh

    /**
     * Variable servant à indiquer le mode déboguage de l'application.
     */
    public static boolean debug = true; // Debug and programming variables

    /**
     * Constructeur principal de la classe.
     * @param filename  (le fichier à exécuter)
     */
    public ExecutorPlato(String filename) {
        this.filename = filename;
    }

    /**
     * Constructeur secondaire de la classe.
     */
    public ExecutorPlato() {
    }

    /**
     * Fonction exécutant une application.
     * Note: the filename must be specified in the constructor
     * @return the output (System.out/.err) of the execution
     */
    public String run(String executable) {
        StringBuilder st = new StringBuilder();
        try {
            exitVal = -99;
            //--Chmod
            Runtime r = Runtime.getRuntime();
            r.exec("chmod +x " + executable);
            ///--Execute the .sh script
            String[] com = new String[5];
            for (int i = 0; i < com.length; i++) {
                com[i] = "";
            }
            //Execute the Makefile_test.command.sh
            com[0] = executable;
            Process p = r.exec(com);
            InputStream stdoutput = p.getInputStream();
            InputStream stderr = p.getErrorStream();
            InputStreamReader isr = new InputStreamReader(stdoutput);
            InputStreamReader isr2 = new InputStreamReader(stderr);
```

```
            BufferedReader br = new BufferedReader(isr);
            BufferedReader br2 = new BufferedReader(isr2);
            String line = null;
            while ((line = br.readLine()) != null) {
                st.append(line + "\n");
            }
            exitVal = p.waitFor();
            if (exitVal != 0) {
                while ((line = br2.readLine()) != null) {
                    st.append(line + "\n");
                }
            }
        } catch (Exception ex) {
            System.out.println(" error unable to execute the test... Error message
            follow:");
            ex.printStackTrace();
            exitVal = -1;
        }
        return st.toString();
    } // End run

    /**
     * Fonction réalisant le test de dépendance en exécutant l'application Cetus.
     * Note: the filename must be specified in the constructor
     * @return the output (System.out/.err) of the execution
     */
    public String run() {
        System.out.println("\n**********************************************");
        System.out.println("*       Computing dependency using Plato1.1... *");
        System.out.println("**********************************************\n");
        if (filename.equals("")) {
            System.out.println("No executable specified.");
            return "No executable specified.";
        }
        StringBuilder st = new StringBuilder();
        try {
            exitVal = -99;
            //--Chmod (just to ensure that the file is executable
            Runtime r = Runtime.getRuntime();
            r.exec("chmod +x " + filename);
            ///--Execute the .sh script
            String[] com = new String[5];
            for (int i = 0; i < com.length; i++) {
                com[i] = "";
            }
            //Execute the Makefile_test.command.sh
            com[0] = filename;
            Process p = r.exec(com);
            InputStream stdoutput = p.getInputStream();
            InputStream stderr = p.getErrorStream();
            InputStreamReader isr = new InputStreamReader(stdoutput);
            InputStreamReader isr2 = new InputStreamReader(stderr);
            BufferedReader br = new BufferedReader(isr);
            BufferedReader br2 = new BufferedReader(isr2);
            String line = null;
            //--System.out
            while ((line = br.readLine()) != null) {
                st.append(line);
                if (debug) {
                    System.out.println(line);
                }
            }
            exitVal = p.waitFor();
            //--System.err
            if (exitVal != 0) {
                while ((line = br2.readLine()) != null) {
                    st.append(line);
                    if (debug) {
                        System.out.println(line);
                    }
                }
            }
        } catch (Exception ex) {
            System.out.println(" error unable to execute the test... Error message,
```

```
            follow:");
            exitVal = -1;
        }
        return st.toString();
    } // End run

    ////////////////////////////////////////////////////////////////////////
    // HELPER FUNCTIONS
    /**
     * Fonction supprimant un fichier.
     * @param filename (to delete)
     * @return true if success
     */
    public boolean deleteFile(String filename) {
        try {
            File outtree = new File(filename);
            if (outtree.exists()) {
                outtree.delete();
            }
        } catch (Exception e) {
            System.out.println("Unable to delete file " + filename);
            return false;
        }
        return true;
    }

    /**
     * Fonction vérifiant la présence d'un fichier.
     * @param filename
     * @return true if file Exists
     */
    public static boolean FileExists(String filename) {
        File f = new File(filename);
        if (f == null || f.isDirectory()) {
            return false;
        }
        return f.exists();
    }
}
```

Tableau E.5 - Le code source de la classe *ExecutorPlato.java*.

Le code source de la classe *GlobalDataStore.java* est présenté dans le Tableau E.6.

```
/* La classe GlobalDataStore.java */

package bouclesablecc;

import java.util.Vector;

/**
 * Classe implémentant une structure contenant les informations
 * sur les différentes boucle 'for' détectées
 * pour analyse des dépendances.
 * @author Alpha Boubacar Diallo
 * @since May 2011
 */
public class GlobalDataStore {
    ////////////////////////////////////////////////////////
    /// Constant

    /**
     * Nombre maximal de boucle à traiter.
     */
    int taille = 100;
    ////////////////////////////////////////////////////////
    /// Variables
    /**
     * Liste des boucles 'for' détectées dans le code source.
     */
    public static Vector<StructureBouclePlato> listeBoucles;

    ////////////////////////////////////////////////////////
    /// Constructor
    /**
     * Constructeur principal.
     */
    public GlobalDataStore() {
        listeBoucles = new Vector(taille);
        for (int i = 0; i < taille; i++) {
            StructureBouclePlato struct = new StructureBouclePlato();
            listeBoucles.add(i, struct);
        }
    }
}
```

Tableau E.6 - Le code source de la classe *GlobalDataStore.java*.

Le code source de la classe *InterpreterCAdder.java* est présenté dans le Tableau E.7.

```
/* La classe InterpreterCAdder.java */

package bouclesablecc;

import simpleCAdder.node.*;
import simpleCAdder.analysis.DepthFirstAdapter;
import java.util.regex.*;

/**
 * Classe principale de la détection des boucles 'for' basée sur le 'parser'
 * généré par SableCC.
 * @author Alpha Boubacar Diallo
 * @since May 2011
 */
public class InterpreterCAdder extends DepthFirstAdapter {

    /**
     * Variable de stockage des boucles.
     */
    static public GlobalDataStore globalDataStore = new GlobalDataStore();
    /**
     * Variable temporaire indiquant l'indice de la boucle traîtée.
     */
    public static int indice = 0;
    /**
     * Variables identifiant l'indice de la boucle 'for'.
     */
    String indices = "";
    /**
     * Variable indiquant le nombre actuel de boucles 'for'.
     */
    public static int count = 0;
    /**
     * Variable temporaire indiquant un changement dans les indices.
     */
    boolean changement = true;
    /**
     * Variable servant à indiquer le mode déboguage de cette classe.
     */
    boolean debug = true;
    /**
     * Variable temporaire indiquant la découverte d'une boucle 'for'.é
     */
    boolean flag_loop = false;

    @Override
    public void caseAForlIterationStatement(AForlIterationStatement node) {
        //--Note: We need to keep the ; in the boucle declaration for this For

        if (changement) {
            String boucle = "for (" + node.getDecl() + ";" + node.getCond()
                    + ";" + node.getIter() + ")" + node.getStatement() + "\n";
            changement = false;
            globalDataStore.listeBoucles.elementAt(indice).boucleFor = boucle;
            Pattern p1 = Pattern.compile("for");
            // Create a matcher with an input string
            String[] tab = p1.split(node.getStatement().toString());
            count = tab.length;
            flag_loop = true;
        }

        System.out.println("For-loop (1) : " + "for (" + node.getDecl()
                + node.getCond() + ";" + node.getIter() + ")" + node.getStatement());

        String r = "";
        Pattern p2 = Pattern.compile(" ");

        // Split input with the pattern
        String[] result = p2.split(node.getDecl().toString());
        r = r + result[0] + "\t" + result[2] + "\t";// i = valeur, indice de valeur = 2
```

```
        // Split input with the pattern
        result = p2.split(node.getCond().toString());
        r = r + result[2] + "\t";           // i < valeur, indice de aleur = 2
        indices = indices + result[0] + "\n";

        result = p2.split(node.getIter().toString());
        if (result[0].equals("++") || result[1].equals("++")) {//i ++, indice du ++ = 1
            r = r + "+1";
        } else if (result[1].equals("+")) {
            r += "+";
            for (int i = 2; i < result.length; i++) {
                if (isNumeric(result[i])) {
                    r = r + result[i];
                } else {
                    System.out.println("Error in the computatuion : " + node.getIter());
                    System.exit(60);
                }
            }
        } else if (result[0].equals("--") || result[1].equals("--")) {
            r = r + "-1";           //i ++, indice du -- = 1
        } else if (result[1].equals("-")) {
            r += "-";
            for (int i = 2; i < result.length; i++) {
                if (isNumeric(result[i])) {
                    r = r + result[i];
                } else {
                    System.out.println("Error in the computatuion : " + node.getIter());
                    System.exit(60);
                }
            }
        } else if (result[1].equals("+=")) {
            if (isNumeric(result[2])) {
                r = r + "+" + result[2];
            } else {
                System.out.println("Error in the computatuion : " + node.getIter());
                System.exit(60);
            }

        } else if (result[1].equals("-=")) {
            if (isNumeric(result[2])) {
                r = r + "+" + result[2];
            } else {
                System.out.println("Error in the computatuion : " + node.getIter());
                System.exit(60);
            }

        } else {
            System.out.print("InterpreterCAdder: iteration error. \n");
            System.out.println("Error in the computatuion : " + node.getIter());
            System.exit(60);
        }
        globalDataStore.listeBoucles.elementAt(indice).inputBouclesFor.addElement(r);
        // Create a pattern to match cat
        Pattern p1 = Pattern.compile("for");
        // Create a matcher with an input string
        Matcher m = p1.matcher(node.getStatement().toString());
        boolean result1 = m.find();
        if (result1) {
            globalDataStore.listeBoucles.elementAt(indice).maxIndice++;
            node.getStatement().apply(this);
            count--;
        } else if (count > 0) {
            node.getStatement().apply(this);
            count--;
        }

        if (count == 0) {
            globalDataStore.listeBoucles.elementAt(indice).inputIndices = indices;
            indices = "";
            indice++;
            changement = true;
            flag_loop = false;
        }
    }
```

```
@Override
public void caseAAssignAssignmentExpression(AAssignAssignmentExpression node) {

    /* Detection of all assignments */
    if (flag_loop == true) {
        if (node.getAssignmentOperator().toString().trim().equals("=") == true) {
            globalDataStore.listeBoucles.elementAt(indice).inputAssignement.
                    addElement(node.getUnaryExpression().toString().trim());
            globalDataStore.listeBoucles.elementAt(indice).outputAssignement.
                    addElement(node.getAssignmentExpression().toString().trim());
        }
       else if (node.getAssignmentOperator().toString().trim().equals("+=")== true){
            String a = node.getUnaryExpression().toString().trim();
            String b = node.getAssignmentExpression().toString().trim();
            String ajout = a + " + " + b;
            globalDataStore.listeBoucles.elementAt(indice).inputAssignement.add(a);
            globalDataStore.listeBoucles.elementAt(indice).
                    outputAssignement.add(ajout);
        }
        else if(node.getAssignmentOperator().toString().trim().equals("-=")==true){
            String a = node.getUnaryExpression().toString().trim();
            String b = node.getAssignmentExpression().toString().trim();
            String ajout = a + " + " + b;
            globalDataStore.listeBoucles.elementAt(indice).inputAssignement.add(a);
            globalDataStore.listeBoucles.elementAt(indice).
                    outputAssignement.add(ajout);
        }
       else if(node.getAssignmentOperator().toString().trim().equals("*=") == true) {
            String a = node.getUnaryExpression().toString().trim();
            String b = node.getAssignmentExpression().toString().trim();
            String ajout = a + " + " + b;
            globalDataStore.listeBoucles.elementAt(indice).inputAssignement.add(a);
            globalDataStore.listeBoucles.elementAt(indice).
                    outputAssignement.add(ajout);
        }
       else if (node.getAssignmentOperator().toString().trim().equals("/=")== true){
            String a = node.getUnaryExpression().toString().trim();
            String b = node.getAssignmentExpression().toString().trim();
            String ajout = a + " + " + b;
            globalDataStore.listeBoucles.elementAt(indice).inputAssignement.add(a);
            globalDataStore.listeBoucles.elementAt(indice).
                    outputAssignement.add(ajout);
        }
    }
}

@Override
public void caseAFor2IterationStatement(AFor2IterationStatement node) {

    if (changement) {
        flag_loop = true;
        String boucle = "for (" + node.getDeclaration() + node.getCond() + ";"
                + node.getIter() + ")" + node.getStatement() + "\n";
        changement = false;
        globalDataStore.listeBoucles.elementAt(indice).boucleFor = boucle;
        Pattern p1 = Pattern.compile("for");
        // Create a matcher with an input string
        String[] tab = p1.split(node.getStatement().toString());
        count = tab.length;
    }
    System.out.println("For-loop (2) : " + "for (" + node.getDeclaration()
            + node.getCond() + ";" + node.getIter() + ")" + node.getStatement());
    String r = "";
    Pattern p2 = Pattern.compile(" ");
    // Split input with the pattern
    String[] result = p2.split(node.getDeclaration().toString());
    result[1] + "\t" + result[3] + "\t";// int i=valeur,indice de valeur = 3
    // Split input with the pattern
    result = p2.split(node.getCond().toString());
    for (int i = 2; i < result.length; i++) {
        r = r + result[i];  // i < valeur, indice de valeur = 2
    }
    r += "\t";
```

```
        indices = indices + result[0] + "\n";

        // Split input with the pattern
        result = p2.split(node.getIter().toString());
        if (result[0].equals("++") || result[1].equals("++")) { //i ++, indice du ++ = 1
            r = r + "+1";
        } else if (result[1].equals("+")) {
            r += "+";
            for (int i = 2; i < result.length; i++) {
                if (isNumeric(result[i])) {
                    r = r + result[i];
                } else {
                    System.out.println("Error in the computatuion : " + node.getIter());
                    System.exit(60);
                }
            }
        } else if (result[0].equals("--") || result[1].equals("--")) {
            r = r + "-1";            //i ++, indice du -- = 1
        } else if (result[1].equals("-")) {
            r += "-";
            for (int i = 2; i < result.length; i++) {
                if (isNumeric(result[i])) {
                    r = r + result[i];
                } else {
                    System.out.println("Error in the computatuion : " + node.getIter());
                    System.exit(60);
                }
            }
        } else if (result[1].equals("+=")) {
            if (isNumeric(result[2])) {
                r = r + "+" + result[2];
            } else {
                System.out.println("Error in the computatuion : " + node.getIter());
                System.exit(60);
            }
        } else if (result[1].equals("-=")) {
            if (isNumeric(result[2])) {
                r = r + "+" + result[2];
            } else {
                System.out.println("Error in the computatuion : " + node.getIter());
                System.exit(60);
            }
        } else {
            System.out.print("InterpreterCAdder: iteration error. \n");
            System.out.println("Error in the computatuion : " + node.getIter());
            System.exit(60);
        }
        if (debug) {
            System.out.println("Return : " + r);
        }
        globalDataStore.listeBoucles.elementAt(indice).inputBouclesFor.addElement(r);

        // Create a pattern to match cat
        Pattern p1 = Pattern.compile("for");
        // Create a matcher with an input string
        Matcher m = p1.matcher(node.getStatement().toString());
        boolean result1 = m.find();
        if (result1) {
            globalDataStore.listeBoucles.elementAt(indice).maxIndice++;
            node.getStatement().apply(this);
            count--;
        }
        else if (count > 0) {
            node.getStatement().apply(this);
            count--;
        }
        if (count == 0) {
            globalDataStore.listeBoucles.elementAt(indice).inputIndices = indices;
            indices = "";
            indice++;
            changement = true;
            flag_loop = false;
        }
    }
```

```
    /**
     * Fonction permettant de vérifier si la chaîne de caractères spécifiée est
     * un nombre.
     * @param s
     * @return true si c'est un nombre.
     */
    private boolean isNumeric(String s) {
        boolean retour = true;
        s = s.trim();
        for (int i = 0; i < s.length(); i++) {
            if (s.charAt(i) >= '0' && s.charAt(i) <= '9') {
            } else {
                retour = false;
            }
        }
        return retour;
    }
}
```

Tableau E.7 - Le code source de la classe *InterpreterCAdder.java*.

Le code source de la classe *InterpreterCInit.java* est présenté dans le Tableau E.8.

```
/* La classe InterpreterCInit.java */

package bouclesablecc;

import simpleCInit.node.*;
import simpleCInit.analysis.DepthFirstAdapter;

/**
 * Classe servant à faire une deuxième uniformisation du code source en entrée.
 * e.g. enlever les espaces, ajouter des sauts de ligne.
 * @author Alpha Boubacar Diallo
 * @since July 2011
 */
public class InterpreterCInit extends DepthFirstAdapter {

    /**
     * Variable temporaire contenant la chaîne de caractères à traiter.
     */
    public String s = "";
    /**
     * Classe servant à enlever les caractères espaces;
     */
    static public BlankRemover blankremover;

    @Override
    public void caseAFile(AFile node) {

        for (int i = 0; i < node.getPart().size(); i++) {
            node.getPart().get(i).apply(this);
        }

    }

    @Override
    public void caseABseolPart(ABseolPart node) {
        s += ""; /*"\nPartbseol\n " + node.getBseol().toString();*/
    }

    @Override
    public void caseABsPart(ABsPart node) {
        s += blankremover.lrtrim(node.getBs().toString());
    }

    @Override
    public void caseAEolPart(AEolPart node) {
        s += "\n";
    }

    @Override
    public void caseATextPart(ATextPart node) {
        s += blankremover.ltrim(node.getText().toString());
    }
}
```

Tableau E.8 - Le code source de la classe *InterpreterCInit.java.java.*

Le code source de la classe *InterpreterCPreprocessing.java* est présenté dans le Tableau E.9.

```
/* La classe InterpreterCPreprocessing.java */
package bouclesablecc;

import simpleCPreprocessing.node.*;
import simpleCPreprocessing.analysis.DepthFirstAdapter;
import java.util.regex.*;

/**
 * Classe permettant d'enlever les parties du code source non nécessaire
 * pour l'analyse, tels que l'instruction '#pragma' et les commentaires.
 * @author Alpha Boubacar Diallo
 * @since July 2011
 */
public class InterpreterCPreprocessing extends DepthFirstAdapter {

    /**
     * Variable temporaire contenant la chaîne de caractères à traiter.
     */
    public String s = "";
    /**
     * Classe servant à enlever les caractères espaces;
     */
    static public BlankRemover blankremover;

    @Override
    public void caseAFile(AFile node) {

        for (int i = 0; i < node.getPart().size(); i++) {
            node.getPart().get(i).apply(this);
        }

    }

    @Override
    public void caseAPretextPart(APretextPart node) {
        s += ""; /*(node.getPreprotext().toString());*/
    }

    @Override
    public void caseATextPart(ATextPart node) {
        s += /*"\nPart text \n " +  */ (node.getText().toString());
    }

    @Override
    public void caseAEolPart(AEolPart node) {
        s += ""; /*"\n";*/
    }
}
```

Tableau E.9 - Le code source de la classe *InterpreterCPreprocessing.java*.

Le code source de la classe *main.java* est présenté dans le Tableau E.10.

```
/* La classe principale main.java */
package bouclesablecc;

import java.io.*;
import java.text.SimpleDateFormat;
import java.util.regex.*;
import java.util.*;

/**
 * Classe principale du paquet bouclesablecc qui détecte les boucles
 * for dans un code source et exécute les différentes applications permettant
 * la détection des dépendances des données.
 * @author Alpha Boubacar Diallo et Etienne Lord
 * @since June 2011
 */
public class main {

    ////////////////////////////////////////////////////////////////////////////
    /// Function class
    /**
     * Classe servant à enlever les caractères espaces;
     */
    static public BlankRemover blankremover;  //-- Remove blank in the input file
    /**
     * Classe servant à exécuter l'application PLATO.
     */
    static public ExecutorPlato executor; //-- Class to execute the dependency test
    /**
     * Classe servant à exécuter l'application Cetus.
     */
    static public ExecutorCetus executorcetus;
    /**
     * Classe servant à générer les 'Makefiles' utilisés par l'application PLATO.
     */
    static public Makefile makefile;                  //-- The makefile
    /**
     * Classe servant à générer les pages web de résultats des différents tests.
     */
    static public WebPage webpage;
    //Helper class to create the results web page (loop transform + dependency detection)
    ////////////////////////////////////////////////////////////////////////////
    /// Variables
    public static String output_path = ""; // Output path
    public static String input_file = ""; // Input file
    public static String testDependance = "0"; // Default 0: LVI
    public static String initPreParsingFile = "outPreParsing.txt";// File before parsing
    // File before preprocessing
    //Results a description of all the loop (for...) analysed.
    public static String initOutFile = "outInit.txt";
    public static String log_file = "log.txt";// Generated outside of this program
    //File after SableCC preprocessing
    public static String preproOutFile = "outPrepro.txt";
    public static String loop_output_file = "loop.txt";
    // Path to the plato directory (/public_html/usagers/plato1.1/)
    public static String path_to_plato = "plato.1.1";
    //Output .cc files to process
    public static String lvi_main_output_file_prefix = "lvitest_main";
    public static String pvi_main_output_file_prefix = "pvitest_main"; //
    public static String rvi_main_output_file_prefix = "rvitest_main"; //
    public static String cetus_command_file = "cetus.command.sh";//File to execute cetus
    public static String makefile_output = "Makefile_test"; //--output file before rename
    public static String loop_dependency_trace = "loop_dependency_trace.txt";
    //--output webpage (dependency detection)
    public static String webpage_output_file = "output.php";
    //--output webpage (loop transformation)
    public static String webpage_output_loop_file = "output_loop.php";


    ////////////////////////////////////////////////////////////////////////////
    /// Flags
    public static boolean debug = true;      //--Debug flag
```

```
public static boolean transformation_only = false; //--Transformation only flag
//////////////////////////////////////////////////////
/// Constants (files required by plato1.1)
public static String lvi_param = "lvi_param_main.cc";
public static String pvi_param = "pvi_param_main.cc";
public static String rvi_param = "rvi_param_main.cc";
public static final SimpleDateFormat dateformat =
        new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

//////////////////////////////////////////////////////////////
/// Error code
/// 0 : Ok!
/// 50: Wrong number of arguments.
/// 51: Unable to find some system files (lvi_param_main...)
/// 52: Bad test. Must be 0, 1, 2 or 3 in the command line
/// 53: -- TO DO --
/// 54: Error in simpleCPreproVisitor
/// 55: Error in simpleCAdderVisitor
/// 56: Unable to create results file (HTML dependency)
/// 57: Unable to create results file (execution of makefile)
/// 58: Error in execution of each cc code...
/// 59: Unable to create results file (HTML loop transform)
/// 60: Unable to compute the loop detected.
/// 99: Parser error
/// 100: No loop found... Probably a parser error... see log
/// 101: Error parsing Cetus
/// 102: Error generating cetus file
/// 103: Unable to create the cetus HTML dependency file
////////////////////////////////////////////////////////////////////////////
/// Main
/**
 * Fonction principale de l'application.
 * @param args
 * @throws IOException
 */
public static void main(String[] args) throws IOException {

    //--Create an instance of the main class
    main m = new main();

    //--Output the program arguments
    System.out.println("**********************************************************");
    System.out.println("**                                                      ");
    System.out.println("**Boucle SableCC - Alpha Boubacar Diallo - February 2012");
    System.out.println("**Dependency test script using Plato v1.1 and SableCC   ");
    System.out.println("**                                                      ");
    System.out.println("**********************************************************");
    System.out.println("**                                                      **");
    System.out.println("**Usage: java -jar BoucleSableCC.jar source.c [test]    **");
    System.out.println("**[outputpath]                                          **");
    System.out.println("**Arguments:                                            **");
    System.out.println("**test :       0: LVI   1: PVI   2: RVI                 **");
    System.out.println("3: Loop transformation 4: Cetus                        **");
    System.out.println("** output_path: directory e.g. usagers/test/            **");
    System.out.println("**********************************************************");

    //--Initialization of the args
    switch (args.length) {
        case 1:
            input_file = args[0];
            break;
        case 2:
            input_file = args[0];
            testDependance = args[1];
            break;
        case 3:
            input_file = args[0];
            testDependance = args[1];
            output_path = args[2];
            if (!output_path.endsWith("/")) {
                output_path += "/";
            }
            break;
        default:
```

```
            System.out.println("Error. Wrong number of arguments.");
            System.exit(50);
            break;
        }

        //--Output some path for debugging
        System.out.println(output_path);
        //--Create the results Webpage container class
        webpage = new WebPage(output_path);

        //--Change if needed some path
        initPreParsingFile = output_path + initPreParsingFile;
        initOutFile = output_path + initOutFile;
        preproOutFile = output_path + preproOutFile;
        loop_output_file = output_path + loop_output_file;
        log_file = output_path + log_file;
        makefile_output = output_path + makefile_output;
        webpage_output_file = output_path + webpage_output_file;
        webpage_output_loop_file = output_path + webpage_output_loop_file;
        loop_dependency_trace = output_path + loop_dependency_trace;

        cetus_command_file = output_path + cetus_command_file;

        //--Clean out
        deleteFile(initPreParsingFile);
        deleteFile(initOutFile);
        deleteFile(preproOutFile);
        deleteFile(loop_output_file);
        deleteFile(log_file);
        deleteFile(makefile_output);
        deleteFile(preproOutFile);
        deleteFile(cetus_command_file);
        deleteFile(loop_dependency_trace);

        //--trex_cluster test
        if (output_path.startsWith("usagers")) {
            lvi_param = "usagers/" + lvi_param;
            pvi_param = "usagers/" + pvi_param;
            rvi_param = "usagers/" + rvi_param;
            path_to_plato = "usagers/" + path_to_plato;
        }

        //--Test if the correct params file are found before we continue...
        if (!(FileExists(lvi_param) && FileExists(pvi_param) && FileExists(rvi_param))) {
            System.out.println("output_path:" + output_path);
            System.out.println(lvi_param + " found " + FileExists(lvi_param));
            System.out.println(pvi_param + " found " + FileExists(pvi_param));
            System.out.println(rvi_param + " found " + FileExists(rvi_param));
            System.out.println("Some required params files are not found...");
            System.out.println("Unable to find some system files (lvi_param_main...)");
            System.exit(51);
        }
        //--Validate the dependence test
        if (!(testDependance.equals("0") || testDependance.equals("1") ||
                testDependance.equals("2") || testDependance.equals("3") ||
                testDependance.equals("4") || testDependance.equals("5"))) {
            System.out.println("Unknown test. Valid dependence tests or transformations "
                    + "are :\n0: Plato LVI\n1: Plato PVI\n2: Plato RVI\n3:Loop "
                    + "Transformation\n4:Cetus Banerjee-Wolfetest\n5:Cetus Range test.");
            System.exit(52);
        } else {
            System.out.println("\n*********************************************");



            if (testDependance.equals("0")) {
                System.out.println("*Computing dependency using LVI test...        *");
            } else if (testDependance.equals("1")) {
                System.out.println("*Computing dependency using PVI test...        *");
            } else if (testDependance.equals("2")) {
                System.out.println("*Computing dependency using RVI test...        *");
            } else if (testDependance.equals("4")) {
                System.out.println("*Computing dependency using Banerjee-Wolfe test*");
            } else if (testDependance.equals("5")) {
```

```
            System.out.println("*Computing dependency using Range test...      *");
        } else {
            System.out.println("*Computing transformation of loop...           *");
        }
        System.out.println("************************************************\n");
    }

    ////////////////////////////////////////////////////////////////////////
    /// Cetus
    if (testDependance.equals("4")) {
        m.runCetus(false);
        System.exit(0);
    }
    if (testDependance.equals("5")) {
        m.runCetus(true);
        System.exit(0);
    }
    //Preprocess the input file to remove any unknown keyword and unknown characters
    //--Added by Etienne Lord - February 2012
    try {
        //-1. Read the file into a buffer
        ArrayList<String> ar = new ArrayList<String>();
        BufferedReader br = new BufferedReader(new FileReader(new File(input_file)));
        while (br.ready()) {
            //-2. Filter the file
            String stri = br.readLine();
            // From ISO C9x  compliant stdint.h for Microsoft Visual Studio
            // Based on ISO/IEC 9899:TC2 Committee draft (May 6, 2005) WG14/N1124
            stri = stri.replaceAll("uint8_t", "unsigned char");
            stri = stri.replaceAll("uint16_t", "unsigned short");
            stri = stri.replaceAll("uint32_t", "unsigned int");
            stri = stri.replaceAll("uint64_t", "unsigned long");
            stri = stri.replaceAll("int8_t", "signed char");
            stri = stri.replaceAll("int16_t", "signed short");
            stri = stri.replaceAll("int32_t", "signed int");
            stri = stri.replaceAll("int64_t", "signed long");
            //--Scan for non standard char
            char replacement = 95;
            String newstri = "";
            for (int i = 0; i < stri.length(); i++) {
                char c = stri.charAt(i);
                if (c > 128) {
                    newstri += replacement;
                } else {
                    newstri += c;
                }
            }
            ar.add(newstri);
        }
        br.close();
        //-3. Write the file back
        PrintWriter pw = new PrintWriter(new FileWriter(new
              File(initPreParsingFile)));
        for (String stri : ar) {
            pw.println(stri);
        }
        pw.flush();
        pw.close();
    } catch (Exception e) {
        System.out.println("Error, unable to read from source file.");
        System.exit(51);
    }
    //--Create the pre-results file
    if (m.simpleCInitVisitor(initPreParsingFile, initOutFile)) {
        if (m.simpleCPreproVisitor(initOutFile, preproOutFile)) {
            m.simpleCAdderVisitor(preproOutFile, testDependance, output_path);
            // CASE 1. If no loop is found...Try to add a foo function to encapsulate
            // the code
            if (webpage.loop.size() == 0) {
                if (debug) {
                    System.out.println("\nNo function found. Encompassing code "
                            + "in a void foo() {...} function.\n");
                }
                try {
```

```
                    BufferedReader br =
                          new BufferedReader(new FileReader(new
                            File(preproOutFile)));
                    String st = "void foo()\n{\n";
                    while (br.ready()) {
                        st += br.readLine() + "\n";
                    }
                    br.close();
                    st += "\n}\n";
                    PrintWriter pw =
                            new PrintWriter(new FileWriter(new File(preproOutFile)));
                    pw.println(st);
                    pw.flush();
                    pw.close();
                    boolean good_parsing =
                            m.simpleCAdderVisitor(preproOutFile, testDependance,
                            output_path);
                    if (!good_parsing) {
                        System.out.println("Warning. Unable to parse the input
                        file");
                        System.exit(99);
                    }
                } catch (Exception e) {
                }
            }
            // CASE 2. We really found nothing. Exit
            if (webpage.loop.size() == 0) {
                System.out.println("No for loop found in input file.");
                System.out.println("Nothing to do. Exiting.");
                System.exit(100);
            } else {
                System.out.println("Found " + webpage.loop.size() + " for loop...");
            }
        } else {
            System.exit(54);
        }
    } else {
        System.exit(55);
    }

    //--No errors... Create the tests if we are not doing a loop transformation
    if (!testDependance.equals("3") && !testDependance.equals("4")) {
        //--Run the dependency check using plato...
        executor = new ExecutorPlato(makefile_output + ".command.sh");
        executor.run();

        //--Execute each test
        if (executor.exitVal == 0) {
            System.out.println("\n***********************************************");
            System.out.println("*            Results of the dependency test.    *");
            System.out.println("***********************************************\n");
            //--Run the test and keep the trace...
            for (String performed_test : makefile.targets_filename) {
                webpage.results.add(executor.run(output_path + "plato.1.1/src/"
                        + performed_test));
                if (executor.exitVal != 0) {
                    System.out.println("Error in executing " + performed_test);
                    String error_results = webpage.results.get(webpage.results.size()
                          - 1);
                    //--Remove the result -> error
                    webpage.results.remove(webpage.results.size() - 1);
                    //--Print to the trace....
                    System.out.println(error_results);
                    //--Add an error code
                    error_results = "Plato error in computing dependence test (error"
                            + executor.exitVal + ")\n" + error_results;
                    webpage.results.add(error_results);
                    //System.exit(58);
                }
            } //--End for performed test...

            //--Create the results file (HMTL)
            if (!webpage.generate(webpage_output_file)) {
                System.out.println("Unable to create results file " +
```

```
                        webpage_output_file);
                    System.exit(56);
                } //--End generate webpage
                //--Error in the test execution
            } else {
                //--Note: debug 21 Juillet 2011 ->
                System.out.println("Unable to generate test files... ");
                System.exit(57);
            } //--End test dependency
            //--Run the loop transformation
        } else if (testDependance.equals("3")) {
            //--Generate the loop-transformation results
            System.out.println("\n*****************************************");
            System.out.println("*            Loop-transformation          *");
            System.out.println("*****************************************\n");
            //--Scan the input file
            VariableData information_on_data = new VariableData();
            try {
                information_on_data.process_file(input_file);
            } catch (Exception e) {
                System.out.println("Error in parsing variables...");
            }

            if (!webpage.generateLoop(webpage_output_loop_file, information_on_data)) {
                System.out.println("Unable to create results file " +
                        webpage_output_loop_file);
                System.exit(59);
            } //--End generate webpage
        }
        //--Normal program termination
        System.out.println("Done.");
        System.exit(0);
        //--Done!
    }

    /**
     * Fonction exécutant l'interprèteur principal sur le code source.
     * @param file
     * @param outfile
     * @return true if the InitVisitor was successfull
     */
    public boolean simpleCInitVisitor(String file, String outfile) {
        try {
            /* Form our AST */
            simpleCInit.lexer.Lexer lexer =
                    new simpleCInit.lexer.Lexer(new PushbackReader(
                    new FileReader(file), 1024));
            simpleCInit.parser.Parser parser = new simpleCInit.parser.Parser(lexer);
            simpleCInit.node.Start ast = parser.parse();
            /* Get our Interpreter going. */
            InterpreterCInit interpInit = new InterpreterCInit();
            ast.apply(interpInit);
            FileWriter fstream = new FileWriter(new File(outfile));
            BufferedWriter out = new BufferedWriter(fstream);
            out.write(interpInit.s);
            out.close();
            fstream.close();
        } catch (Exception e) {
            System.out.println("Error in the InitLexer.");
            System.out.println(e);
            return false;
        }
        return true;
    }

    /**
     * Fonction réalisant un premier nettoyage du fichier source (code).
     * @param file
     * @param outfile
     * @return
     */
    public boolean simpleCPreproVisitor(String file, String outfile) {
        try {
            /* Form our AST */
```

```
        simpleCPreprocessing.lexer.Lexer lexer =
                new simpleCPreprocessing.lexer.Lexer(new PushbackReader(
                new FileReader(file), 1024));
        simpleCPreprocessing.parser.Parser parser =
                new simpleCPreprocessing.parser.Parser(lexer);
        simpleCPreprocessing.node.Start ast = parser.parse();
        /* Get our Interpreter going. */
        InterpreterCPreprocessing interpPrepro = new InterpreterCPreprocessing();
        ast.apply(interpPrepro);

        FileWriter fstream = new FileWriter(outfile);
        BufferedWriter out = new BufferedWriter(fstream);
        out.write(interpPrepro.s);
        out.close();
        fstream.close();
    } catch (Exception e) {
        System.out.println("Error in PreproVisitor.");
        System.out.println(e);
        return false;
    }
    return true;
}

/**
 * Fonction principale analysant les boucles 'for' à l'aide de l'application Plato
 * v1.1
 * @param file
 * @param testDependance
 * @param path
 * @return True, if no parsing error, False otherwise
 * @throws IOException
 */
public boolean simpleCAdderVisitor(String file, String testDependance, String path)
        throws IOException {
    boolean antidep = false;
    //--Note: Change the Makefile to the proper output PATH specified at
    //  the initialization (args)
    makefile = new Makefile(makefile_output, output_path, path_to_plato);

    /* On passe le lexer */
    try {
        /* Form our AST */
        simpleCAdder.lexer.Lexer lexer = new simpleCAdder.lexer.Lexer(new
        PushbackReader(new FileReader(file), 1024));
        simpleCAdder.parser.Parser parser = new simpleCAdder.parser.Parser(lexer);
        simpleCAdder.node.Start ast = parser.parse();

        /* Get our Interpreter going. */
        InterpreterCAdder interp = new InterpreterCAdder();
        ast.apply(interp);
    } catch (Exception e) {
        System.out.println(e);

        //--Look here for parser error...
        String error_msg = e.getMessage();
        Integer line = Integer.valueOf(error_msg.substring(1,
        error_msg.indexOf(",")));
        Integer row = Integer.valueOf(error_msg.substring(error_msg.indexOf(",")
                + 1, error_msg.indexOf("]")));
        BufferedReader br = new BufferedReader(new FileReader(new File(file)));
        int count_line = 0;
        String stri = "";
        while (br.ready() && count_line < line) {
            stri = br.readLine();
            count_line++;
        }
        br.close();
        System.out.println("Line (" + line + "):" + stri);
        String unknown_word = stri.substring(row - 1, stri.indexOf(" ", row));
        System.out.println("Keyword causing error: " + unknown_word);
        //System.out.println(line+":"+row);  <
        return false; //--Parsing error
    }
```

```
/* Boucles à traiter*/
FileWriter fstream = new FileWriter(loop_output_file);
BufferedWriter outBoucle = new BufferedWriter(fstream);

for (int i = 0; i < InterpreterCAdder.indice; i++) {

    String boucle = InterpreterCAdder.globalDataStore.listeBoucles.
            elementAt(i).boucleFor;
    //--Output to file the loop
    outBoucle.write(boucle);
    //--Add to webpage
    WebPage.loop.add(boucle);

    String progDependances = "";
    String addSymbol = "addSymbolicVariable(";
    String addCommon = "addCommonLoopVariable(";
    String addSub = "addSubscript(";
    String var = "Variable ";
    String var2 = ": ";
    String var3 = "return ";

    //--Array to know if we already added or not the variables or not the
    // variable...
    //--The key is the variable, the value is the number...
    //--Note: the number is only for statistic...

    //-What Variables where added
    ArrayList<String> addedProgDependances = new ArrayList<String>();
    //-What Variables where added
    ArrayList<String> addedVariables = new ArrayList<String>();

    Pattern p = Pattern.compile("\\n");
    // Split input with the pattern
     /* On récupère les indices utilisés dans les boucles */
    String[] indicesTab =
    p.split(GlobalDataStore.listeBoucles.elementAt(i).inputIndices);
    for (int j = 0; j < indicesTab.length; j++) {
        // CASE 1. We don't have the variables...
        if (!addedVariables.contains(indicesTab[j].trim())) {
            addedVariables.add(indicesTab[j].trim());
        }
    }

    int maxIndice = GlobalDataStore.listeBoucles.elementAt(i).maxIndice;
    var3 += maxIndice + ";\n";

    /* Récupération des variables de boucles pour le programme de
     test des dépendances : addCommonLoopVariable*/
    for (int j = 0; j < maxIndice; j++) {
        String loopVariable = GlobalDataStore.listeBoucles.elementAt(i).
                inputBouclesFor.elementAt(j).toString().replace("\t", ",");
        String progDep = addCommon + loopVariable + ");";
        if (!addedProgDependances.contains(progDep)) {
            addedProgDependances.add(progDep);
            System.out.println("Variable(s) in for-loop: " + loopVariable);
        }
    }

    ////////////////////////////////////////////////////////////////////
    //--Current loop struct
    webpage.current_loop_struct.clear();
    webpage.current_loop_struct.addLoop(boucle);

    /* Recherche des variables pouvant être dépendantes */
    for (int j = 0; j < GlobalDataStore.listeBoucles.elementAt(i).
            inputAssignement.size(); j++) {
        antidep = false;
        String in = GlobalDataStore.listeBoucles.elementAt(i).inputAssignement.
                elementAt(j).toString().replace(" ", "");  // input variable
        String out = outputAssignement(GlobalDataStore.listeBoucles.elementAt(i)
                .outputAssignement.elementAt(j).toString().replace(" ", ""));

        String[] indices =
        p.split(GlobalDataStore.listeBoucles.elementAt(i).inputIndices);
```

```
String[] outTab = p.split(out);

////////////////////////////////////////////////////////////////
/// Add loop variables for (loop transform
webpage.current_loop_struct.addIn(inputAssignement(in));
webpage.current_loop_struct.addSync(indices[0]);
System.out.println(" addIn : " + inputAssignement(in));
System.out.println(" addSync : " + indices[0]);
//--Iterate over all output
for (int k = 0; k < outTab.length; k++) {
    if (ajouterVariable(inputAssignement(outTab[k]), indices)) {
        webpage.current_loop_struct.addOut(inputAssignement(outTab[k]));
        if (inputAssignement(in).equals(inputAssignement(outTab[k]))) {
            /* Recherches des indices entrées et en sortie */
            String[] indicesIn = p.split(indices(in));
            String[] indicesOut = p.split(indices(outTab[k].toString()));
            Pattern p1 = Pattern.compile("[\\+\\-\\*\\/]");
            /* Récupération des variables input de boucles pour le
             programme de test des dépendances : addSubscript*/
            for (int l = 0; l < indicesIn.length; l++) {
                // Split input with the pattern
                /* On récupère les indices utilisés dans les boucles */
                String[] tab = p1.split(indicesIn[l]);
                for (int m = 0; m < tab.length; m++) {
                    if (debug) {
                        System.out.println("Adding (0): " +
                        tab[m].trim());
                    }
                    if (ajouterVariable(tab[m].trim(), indicesTab) ==
                    true) {
                        if (!addedVariables.contains(tab[m].trim())) {
                            if (debug) {
                                System.out.println("Adding (0 - true): "
                                + tab[m]);
                            }
                            addedVariables.add(tab[m].trim());
                        }
                        String progDep = addSymbol + tab[m].trim() +
                        ");";
                        if (!addedProgDependances.contains(progDep)) {
                            addedProgDependances.add(progDep);
                            testDependance = "1";
                        }
                    }
                }
            }

            /* Récupération des variables Ouput de boucles pour
             le programme de test des dépendances : addSubscript*/
            for (int l = 0; l < indicesOut.length; l++) {
                // Split input with the pattern
                /* On récupère les indices utilisés dans les boucles */
                String[] tab = p1.split(indicesOut[l]);
                for (int m = 0; m < tab.length; m++) {
                    if (debug) {
                        System.out.println("Adding (1): " + tab[m]);
                    }
                    if (ajouterVariable(tab[m].trim(), indicesTab) ==
                    true) {
                        if (!addedVariables.contains(tab[m].trim())) {
                            if (debug) {
                                System.out.println("Adding (1): " +
                                tab[m].trim());
                            }
                            addedVariables.add(tab[m].trim());
                        }
                        String progDep = addSymbol + tab[m].trim() +
                        ");";
                        if (!addedProgDependances.contains(progDep)) {
                            addedProgDependances.add(progDep);
                            testDependance = "1";
                        }
                    }
                }
```

```
                        try {
                            String progSub = addSub + indicesOut[l] + " , "
                                    + indicesIn[l] + ");";
                            if (!addedProgDependances.contains(progSub)) {
                                addedProgDependances.add(progSub);
                            }
                        } catch (Exception e) {
                            System.out.println("** (Warning) Out:" +
                            indicesOut.length +
                           " In:" + indicesIn.length + " while addSubscript **");
                            for (String s : indicesOut) {
                                System.out.println("Out:" + s);
                            }
                            for (String s : indicesIn) {
                                System.out.println("In:" + s);
                            }
                        }
                    }
                } //--End if outTab
            } //--For outTab
        } //--If indice
    } //--End GlobalDataStore.listeBoucles

    Vector inputs = new Vector();
    Vector outputs = new Vector();
    for (int j = 0; j < GlobalDataStore.listeBoucles.elementAt(i).
            inputAssignement.size(); j++) {
        inputs.clear();
        outputs.clear();
        String in = GlobalDataStore.listeBoucles.elementAt(i).inputAssignement.
                elementAt(j).toString().replace(" ", "");  // input variable
        inputs.add(in);
        for (int k = j + 1; k < GlobalDataStore.listeBoucles.elementAt(i).
                inputAssignement.size(); k++) {
            String out =
            outputAssignement(GlobalDataStore.listeBoucles.elementAt(i).
                    outputAssignement.elementAt(k).toString().replace(" ", ""));
            String[] temp = out.split("\n");
            for (int l = 0; l < temp.length; l++) {
                outputs.add(temp[l].trim());
            }
        }
        antidep = antidep || antidependencies(inputs, outputs);
    }
    webpage.current_loop_struct.antidependence = antidep;
    // Si la boucle ne contient aucune assignation
    // Create a pattern to match cat
    Pattern p2 = Pattern.compile("addSubscript");
    // Create a matcher with an input string
    String dummy_progDependances = "";
    for (String st : addedProgDependances) {
        dummy_progDependances += st;
    }
    Matcher mat = p2.matcher(dummy_progDependances);
    boolean result = mat.find();
    // Condition is suffisant if result == false -->
    // empty loop or not subscript condition in the test
    if (result == false) {
        // Split input with the pattern
        /* On récupère les indices utilisés dans les boucles */
        Pattern p3 = Pattern.compile(",");
        String loopVariable = "";
        loopVariable =
                GlobalDataStore.listeBoucles.elementAt(i).inputBouclesFor.
                elementAt(0).toString().replace("\t", ",");
        String[] tab = p3.split(loopVariable);
        String progSub = addSub + tab[1] + " , " + tab[2] + ");";
        addedProgDependances.add(progSub);
    }
    WebPage.loop_transform.add(webpage.current_loop_struct.clone());
    Pattern p4 = Pattern.compile("[\\[\\],\\t]");
    String[] tab =
            p4.split(GlobalDataStore.listeBoucles.elementAt(i).
            inputBouclesFor.toString());
```

```
for (int m = 0; m < tab.length; m++) {
    if (debug) {
        System.out.println("Adding (2): " + tab[m]);
    }
    if (ajouterVariable(tab[m], indicesTab) == true) {
        if (!addedVariables.contains(tab[m])) {
            addedVariables.add(tab[m]);
            if (debug) {
                System.out.println("Adding (2 - true): " + tab[m]);
            }
        }
        String progDep = addSymbol + tab[m] + ");";
        if (!addedProgDependances.contains(progDep)) {
            addedProgDependances.add(progDep);
            testDependance = "1";
        }
    }
}

//--This is the symbolic variable that need to be
String stars = "*************************************************************";
System.out.println(stars);
String s = String.format("*Plato initialization - Creating dependency
files*", i + 1);
System.out.println(s);
System.out.println(stars);
//--Compute all the variable to add...
for (String v : addedVariables) {
    var += v + ",";
    var2 += v + '(' + '"' + v + '"' + ')' + ",";
}
var = var.substring(0, var.length() - 1) + ";";
var2 = var2.substring(0, var2.length() - 1);
//--Order (Note: order for the loop is important):
//        e.g for a(...) for b(...) != for b(...) for a(...)
for (String pr : addedProgDependances) {
    if (pr.startsWith("addSymbolicVariable")) {
        progDependances += pr + "\n";
    }
}   ,

for (String pr : addedProgDependances) {
    if (pr.startsWith("addCommonLoopVariable")) {
        progDependances += pr + "\n";
    }
}
for (String pr : addedProgDependances) {
    if (pr.startsWith("addSubscript")) {
        progDependances += pr + "\n";
    }
}
//--Print Out
System.out.println(progDependances);
if (debug) {
    System.out.println("var:" + var);
}
if (debug) {
    System.out.println("var2" + var2);
}
if (debug) {
    System.out.println("var3:" + var3);
}
System.out.println(stars);
//--Creation des fichiers
if (testDependance.equals("0")) {
    insererTestsFichier(lvi_param, output_path + lvi_main_output_file_prefix
            + i + ".cc", var, var2, progDependances, var3);
    makefile.addTarget(lvi_main_output_file_prefix + i);
} else if (testDependance.equals("1")) {
    insererTestsFichier(pvi_param, output_path + pvi_main_output_file_prefix
            + i + ".cc", var, var2, progDependances, var3);
    makefile.addTarget(pvi_main_output_file_prefix + i);
} else if (testDependance.equals("2")) {
    insererTestsFichier(rvi_param, output_path + rvi_main_output_file_prefix
```

```
                    + i + ".cc", var, var2, progDependances, var3);
                makefile.addTarget(rvi_main_output_file_prefix + i);
            }
        }
        outBoucle.close();
        //--Creation des fichiers de sortie
        try {
            //--Makefile
            BufferedWriter makefile_output1 =
                    new BufferedWriter(new FileWriter(new File(makefile.filename)));
            makefile_output1.append(makefile.getOutput());
            makefile_output1.flush();
            makefile_output1.close();

            //--Makefile.command.sh 00> Fichier a executer
            BufferedWriter makefile_output2 =
                    new BufferedWriter(new FileWriter(new File(makefile.filename +
                    ".command.sh")));
            makefile_output2.append(makefile.getCommand());
            makefile_output2.flush();
            makefile_output2.close();
        } catch (Exception e) {
            System.out.println("Unable to create Makefile "
                    + makefile.filename + ". Error message follow:");
            e.printStackTrace();
        }
        return true; //--No parsing error...
    } //--End

    /**
     * Fonction exécutant la détection de dépendance des boucles 'for' à l'aide de
     * l'application Cetus.
     * @param RangeTest if false, we will performed the Banerjee-Wolfe,
     * otherwise the Range test.
     */
    public void runCetus(boolean RangeTest) {
        //--1. Create the command file (.sh) use to run the program
        executorcetus =
                new ExecutorCetus(cetus_command_file, output_path, initPreParsingFile);
        //--Set the test to run (see above)
        executorcetus.range_test = RangeTest;
        executorcetus.createCommandFile();

        //--2. Filter the input file to avoid bad characters
        try {
            //-2.1. Read the file into a buffer
            ArrayList<String> ar = new ArrayList<String>();
            BufferedReader br = new BufferedReader(new FileReader(new File(input_file)));
            while (br.ready()) {
                //-2.2. Filter the file
                String stri = br.readLine();
                //--Scan for non standard char
                char replacement = 95;
                String newstri = "";
                for (int i = 0; i < stri.length(); i++) {
                    char c = stri.charAt(i);
                    if (c > 128) {
                        newstri += replacement;
                    } else {
                        newstri += c;
                    }
                }
                ar.add(newstri);
            }
            br.close();

            //-2.3. Write the file back
            PrintWriter pw = new PrintWriter(new FileWriter(new
            File(initPreParsingFile)));
            for (String stri : ar) {
                pw.println(stri);
            }
            pw.println("\n"); //--Add extra for security with Cetus
            pw.flush();
```

```
            pw.close();

        } catch (Exception e) {
            System.out.println("Error, unable to read from source file.");
            System.exit(51);
        }

        //--3. Run Cetus and kept the treace log in (results)
        StringBuilder results = executorcetus.run();

        System.out.println("\n*************************************************");
        System.out.println("*       Cetus 1.3 Analysis...                   **");
        System.out.println("*************************************************\n");

        //--4. Analyse the output file
        try {
            PrintWriter pw = new PrintWriter(new FileWriter(new
            File(loop_dependency_trace)));
            pw.println("*****************************************************");
            pw.println("** Cetus trace - " + returnCurrentDateAndTime());
            pw.println("** See Cetus webpage for detailed informations.");
            pw.println("** http://cetus.ecn.purdue.edu");
            pw.println("*****************************************************");
            pw.println("Cetus output file:");
            ArrayList<StructureBoucleCetus> boucles = new
            ArrayList<StructureBoucleCetus>();
            if (FileExists(output_path + "cetus_output/outPreParsing.txt")) {
                //--1. Parse Cetus output file
                try {
                    BufferedReader br =
                            new BufferedReader(new FileReader(new File(output_path
                            + "cetus_output/outPreParsing.txt")));
                    StructureBoucleCetus tmp = new StructureBoucleCetus();
                    boolean inBoucle = false; //--Flag to know if we are in a cetus loop

                    while (br.ready()) {
                        String stri = br.readLine();
                        pw.println(stri);
                        if (stri.startsWith("#pragma cetus private")) {
                            //--Extract the private variables
                            String privateVar = stri.substring(stri.indexOf('(')
                                    + 1, stri.indexOf(')'));
                            tmp.loop_private = privateVar;
                        } else if (stri.startsWith("#pragma loop name")) {
                            String tmp_name = stri.substring(18).trim();
                            if (!tmp.loop_name.equals(tmp_name)) {
                                if (!tmp.loop_name.isEmpty()) {
                                    tmp.loop_for = cleanString(tmp.loop_for);
                                    boucles.add(tmp);
                                    tmp = new StructureBoucleCetus();
                                }
                                tmp.loop_name = tmp_name;
                                //--Compute level (used in the webpage desing)
                                Pattern p_ends = Pattern.compile("(#[0-9]*)");
                                Matcher m_ends = p_ends.matcher(tmp.loop_name);
                                while (m_ends.find()) {
                                    tmp.level++;
                                }
                                //--Set the boucle flag
                                inBoucle = true;
                            }
                        } else if (stri.startsWith("#pragma cetus parallel")) {
                            //--DO nothing
                        } else if (inBoucle) {
                            //System.out.println(stri);
                            if (!stri.startsWith("#")) {
                                //--Calculate indent level
                                for (char c : stri.toCharArray()) {
                                    if (c == '{') {
                                        tmp.indentlevel++;
                                    }
                                    if (c == '}') {
                                        tmp.indentlevel--;
                                    }
```

```
                    }
                    tmp.loop_for += stri + "";
                }
            }
        } //--End while
        //--Add the last boucle
        br.close();
        //--Last clean up
        tmp.loop_for = cleanString(tmp.loop_for);
        boucles.add(tmp);
        System.out.println("Total loops found by Cetus: " + boucles.size());
        pw.println("\n***********************************************");
        pw.println("Total loops found by Cetus: " + boucles.size());
        for (int j = 0; j < boucles.size(); j++) {
            StructureBoucleCetus tmp_boucle = boucles.get(j);
            pw.println("Loop " + (j + 1) + ": " + tmp_boucle.loop_name);
            pw.println(tmp_boucle.loop_for);
        }
    } catch (Exception e) {
        e.printStackTrace();
    }

    //-5. Parse Cetus output...Scan the results
    try {
        String[] str_results = results.toString().split("\n");
        StructureBoucleCetus tmp = new StructureBoucleCetus();//Tmp structure
        pw.println("**************************************");
        pw.println("* Cetus dependency identification... ");
        pw.println("**************************************");
        for (String stri : str_results) {
            if (stri.startsWith("[AUTOPAR]")) {
                stri = stri.substring(10).trim();
                pw.println(stri);
                if (stri.startsWith("Loop is named")) {
                    String tmp_name = stri.substring(14);
                    if (!tmp.loop_name.equals(tmp_name)) {
                        //--Replace the old tmp
                        for (int i = 0; i < boucles.size(); i++) {
                           if(boucles.get(i).loop_name.equals
                           (tmp.loop_name)) {
                                boucles.remove(i);
                                boucles.add(tmp);
                            }
                        }
                        //--Find and put the info into the right nodes
                        for (int i = 0; i < boucles.size(); i++) {
                            if (boucles.get(i).loop_name.equals(tmp_name)) {
                                tmp = boucles.get(i);
                            }
                        }
                    }
                } else if (stri.startsWith("is parallel")) {
                    tmp.parallel = true;
                } else if (stri.startsWith("is serial")) {
                    tmp.serial = true;
                } else if (stri.startsWith("contains array dependences on")){
                    String privateVar =
                            stri.substring(stri.indexOf('{') + 1,
                            stri.indexOf('}'));
                    tmp.loop_dependence += privateVar + " ";
                    tmp.array_dependence += privateVar;
                    tmp.dependence_found = true;
                }
                else if (stri.startsWith("contains scalar dependences on")) {
                    String privateVar =
                            stri.substring(stri.indexOf('{') + 1,
                            stri.indexOf('}'));
                    tmp.loop_dependence += privateVar + " ";
                    tmp.scalar_dependence += privateVar;
                    tmp.dependence_found = true;
                } //--End if
            } //--End AutoPAR
        } //--End parsing results
        //--Final Replace the old tmp
```

```
                for (int i = 0; i < boucles.size(); i++) {
                    if (boucles.get(i).loop_name.equals(tmp.loop_name)) {
                        boucles.remove(i);
                        boucles.add(tmp);
                    }
                }
                pw.flush();
                pw.close();
                //--6. Generate the results web page....
                if (!webpage.generateCetus(webpage_output_file, boucles)) {
                    System.out.println("Unable to create results file "
                            + webpage_output_file);
                    System.exit(103);
                } //--End generate webpage
            } catch (Exception e) {
                e.printStackTrace();
            }
        } else {
            System.out.println("Error: unable to find Cetus output file.");
            System.exit(102);
        }
    } catch (Exception e2) {
        e2.printStackTrace();
    }
}
private boolean antidependencies(Vector inputs, Vector outputs) {
    boolean retour = false;
    for (int i = 0; i < inputs.size(); i++) {
        String in = inputAssignement(inputs.elementAt(i).toString());
        for (int j = 0; j < outputs.size(); j++) {
            String out = inputAssignement(outputs.elementAt(j).toString());
            if (in.equals(out)) {
                if (webpage.current_loop_struct.antidep.contains(in)) {
                } else {
                    webpage.current_loop_struct.antidep.add(in);
                }
                retour = retour || true;
            }
        }
    }
    return retour;
}

/* Nom de la variable input sans les indices et les [] si ces derniers sont présents */
private String inputAssignement(String input) {
    int nb = 0;
    String r = "";
    for (int i = 0; i < input.length(); i++) {
        if (input.charAt(i) == '[') {
            nb++;
        }
    }
    if (nb > 0) {
        Pattern p5 = Pattern.compile("\\[");
        String[] result = p5.split(input);
        r = result[0];
    } else {
        r = input;
    }
    return r;
}

/**
 * Décomposition du outputAssignement à partir des opérations additives
 * @param output
 * @return
 */
private String outputAssignement(String output) {
    int nb = 0;
    int start = 0;
    String r = "";
    //System.out.println("Sortie : " + output);
    for (int i = 0; i < output.length(); i++) {
        // System.out.println("caractere : " + output.charAt(i));
```

```
        if (output.charAt(i) == '[') {
            nb++;
        } else if (output.charAt(i) == ']') {
            nb--;
        } else if (output.charAt(i) == '+' || output.charAt(i) == '*'
                || output.charAt(i) == '-' || output.charAt(i) == '/') {
            if (nb == 0) {
                r = r + output.substring(start, i) + "\n";
                start = i + 1;
                // System.out.println("string = " + r );
            }
        }
        if (i == output.length() - 1) {
            if (nb == 0) {
                r = r + output.substring(start, i + 1);
                // System.out.println("string = " + r );
            }
        }
    }
    //--remove parenthesis from output
    r = r.replace('(', ' ').replace(')', ' ');
    return r;
}


/**
 * Recherche des indices des variables dépendantes
 * @param in
 * @return
 */
private String indices(String in) {
    int nb = -1;
    int start = 0;
    String r = "";
    for (int i = 0; i < in.length(); i++) {
        /* On doit cherche le caractère [ car les indices se
         trouvent à après ce caractère */
        if (in.charAt(i) == '[') {
            nb = 1;
            start = i + 1;
        } /* On doit cherche le caractère ]  car les indices se
         trouvent à avant ce caractère */
        else if (in.charAt(i) == ']') {
            nb--;
        }
        /* Les indices se trouvent entre le caractère [ et le caractère ]*/
        if (nb == 0) {
            if (start < i) {
                r = r + in.substring(start, i) + "\n";
            } else {
                r = r + "1\n";
            }
            nb = -1;
        }
        /* Sinon 'l'indice est 1 */
        else if ((nb == -1) && (i == in.length() - 1)) {
            r = r + "1\n";
        }
    }
    return r;
}

/**
 * Fonction qui remplace les premières valeurs des indices contenues dans
 * le tableau 'in' par ceux du tableau 'out', pour la chaîne de caractères 's';
 * @param s
 * @param in
 * @param out
 * @param nb
 * @return
 */
public String replaceIndices(String s, String[] in, String[] out, int nb) {
    for (int k = 0; k <= nb; k++) {
        s = s.replace(in[k], out[k]);
```

```
        }
        return s;
    }

    private void insererTestsFichier(String infile, String outfile, String substitution1,
            String substitution2, String substitution3, String substitution4) {
        File file = new File(infile);
        FileInputStream fis = null;
        BufferedInputStream bis = null;
        DataInputStream dis = null;
        String retour = "";
        int insertion = 0;
        try {
            fis = new FileInputStream(file);
            bis = new BufferedInputStream(fis);
            dis = new DataInputStream(bis);
            while (dis.available() != 0) {
                String line = dis.readLine();
                Pattern p = Pattern.compile("__insertion__");
                // Create a matcher with an input string
                Matcher m = p.matcher(line);
                boolean result = m.find();
                if (result) {
                    if (insertion == 0) {
                        retour += substitution1 + "\n";
                        insertion++;
                    } else if (insertion == 1) {
                        retour += substitution2 + "\n";
                        insertion++;
                    } else if (insertion == 2) {
                        retour += substitution3 + "\n";
                        insertion++;
                    } else {
                        retour += substitution4 + "\n";
                        insertion++;
                    }
                } else {
                    retour += line + "\n";
                }
            }
            fis.close();
            bis.close();
            dis.close();
            FileWriter fstream = new FileWriter(outfile);
            BufferedWriter out = new BufferedWriter(fstream);
            ;
            out.write(retour);
            out.close();
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    /**
     * Fonction enlevant de la chaîne de caractères 'toClean' les commentaires.
     * @param toClean
     * @return la nouvelle chaîne de caractères.
     */
    public String cleanString(String toClean) {
        String tmp = toClean;
        System.out.println(toClean);
        try {
            Pattern p_comment = Pattern.compile("(/[*](.*?)[*]/)");
            Matcher m_comment = p_comment.matcher(toClean);
            while (m_comment.find()) {
                tmp = tmp.replace(m_comment.group(1), "");
            }

        } catch (Exception e) {
            e.printStackTrace();
        }
        return tmp;
```

```
    }

    /**
     * Fonction remplaçant les premières valeurs contenues dans le tableau 'in'
     * par ceux du tableau 'out', pour la chaîne de caractères 's';
     * @param in String...
     * @param tab
     * @return true
     */
    public boolean ajouterVariable(String in, String[] tab) {
        //System.out.println("\n in = " + in);
        boolean isNumeric = true;
        boolean ajouter = true;
        in = in.trim();
        for (int i = 0; i < in.length(); i++) {
            if (in.charAt(i) >= '0' && in.charAt(i) <= '9') {
            } else {
                isNumeric = false;
            }
            if (in.charAt(0) == '-' || in.charAt(0) == '+') {
                isNumeric = true;
            }
        }
        if (isNumeric == false) {
            for (int j = 0; j < tab.length; j++) {
                if (tab[j].compareTo(in) == 0) {
                    ajouter = false;
                }
            }
        } else {
            ajouter = false;
        }
        return ajouter;
    }

     * Fonction vérifiant l'existance d'un fichier.
     * @param filename
     * @return true if file Exists
     */
    public static boolean FileExists(String filename) {
        File f = new File(filename);
        if (f == null || f.isDirectory()) {
            return false;
        }
        return f.exists();
    }

    /**
     * Fonction vérifiant l'existance d'un répertoire.
     * @param filename
     * @return true if directory exists
     */
    public static boolean DirExists(String filename) {
        File f = new File(filename);
        if (f == null || !f.isDirectory()) {
            return false;
        }
        return f.exists();
    }

    /**
     * Fonction supprimant un fichier.
     * @param filename (to delete)
     * @return true if success
     */
    public static boolean deleteFile(String filename) {
        try {
            File outtree = new File(filename);
            if (outtree.exists()) {
                outtree.delete();
            }
        } catch (Exception e) {
            return false;
        }
```

```
        return true;
    }
    /**
     * Fonction retournant la date et l'heure courante.
     * @return s A String of the current Date and Time
     */
    public static String returnCurrentDateAndTime() {
        Calendar today = Calendar.getInstance();
        return dateformat.format(today.getTime());
    }
} //--End main class
```

Tableau E.9 - Le code source de la classe *main.java*.

Le code source de la classe *Makefile.java* est présenté dans le Tableau E.11.

```
/*La classe Makefile.java */

package bouclesablecc;
import java.util.ArrayList;

/**
 * Classe permettant la génération des 'Makefile' utilisés
 * lors de la détection des dépendances des données par PLATO.
 * @author Alpha Boubacar Diallo et Etienne Lord
 * --Added TAB in makefile - 30 June 2011 - Etienne
 * @since June 2011
 */
public class Makefile {

    ////////////////////////////////////////////////////////////////////////////
    /// Variables
    /**
     * Variable indiquant le nom de sortie du 'Makefile' généré.
     */
    public String filename = "_Makefile";                          //--Output filename
    /**
     * Liste contenant la liste des fichiers devant être inclus dans ce 'Makefile'.
     */
    ArrayList<String> targets_filename = new ArrayList<String>(); //--Target
    /**
     * Variable contenant le répertoire de sortie du 'Makefile'.
     */
    private String path = "";                                      //output_path
    /**
     * Variable contenant le chemin vers l'application PLATO.
     */
    private String plato_path = "";                                //plato_path
    ////////////////////////////////////////////////////////////////////////////
    /// Constants
    /**
     * Constante représentant une chaîne de caractères (début du 'Makefile')
     */
    String start = "CC = /opt/gcc-2.95.3/bin/g++\n"
            + "AR = ar\n"
            + "CFLAGS = -Wall -O1 -Wno-deprecated\n"
            + "LIBS = $(TARGET1)\n"
            + "INCLUDE = ../include\n"
            + "SOURCE = ../src\n"
            + "OBJECTS = ArithmeticExpression.o \\\n"
            + "BoundedExpression.o Graph.o \\\n"
            + "VIExpression.o \\\n"
            + "LinearExpression.o \\\n"
            + "PolynomialExpression.o \\\n"
            + "RationalExpression.o \\\n"
            + "VILinearExpression.o \\\n"
            + "VIPolynomialExpression.o \\\n"
            + "VIRationalExpression.o \\\n"
            + "BanerjeeTestProblem.o \\\n"
            + "ITest.o ITestProblem.o \\\n"
            + "LinearVITest.o LinearVITestProblem.o \\\n"
            + "PolynomialVITest.o PolynomialVITestProblem.o \\\n"
            + "RationalVITest.o RationalVITestProblem.o \\\n"
            + "MonotonicityCache.o \\\n"
            + "MathUtils.o DDTesting.o Clock.o\n\n"
            + "TARGET1 = ../lib/libplato.a\n";
    //--Target
    //TARGET2 = lvi_test
    //TARGET3 = pvi_test
    //TARGET4 = rvi_test
    //--Note: not used since dynamically created String
    // target="TARGETS = $(TARGET1) $(TARGET2) $(TARGET3) $(TARGET4)\n";


    /**
     * Constante représentant une chaîne de caractères ('o' du 'Makefile')
     */
```

```
String o = "%.o: ../src/%.cc\n\t $(CC) $(CFLAGS) -I$(INCLUDE) -c $<\n";
/**
 * Constante représentant une chaîne de caractères ('all' du 'Makefile')
 */
String all = "all: $(TARGETS)\n\n$(TARGET1) : $(OBJECTS)\n\t$(AR) r $(TARGET1)
             $(OBJECTS)\n";
//$(TARGET2) : $(TARGET1) lvitest_main.o
//        $(CC) $(CFLAGS) -I$(INCLUDE) -o $(TARGET2) lvitest_main.o $(LIBS)
//
//$(TARGET3) : $(TARGET1) pvitest_main.o
//        $(CC) $(CFLAGS) -I$(INCLUDE) -o $(TARGET3) pvitest_main.o $(LIBS)
//
//$(TARGET4) : $(TARGET1) rvitest_main.o
//        $(CC) $(CFLAGS) -I$(INCLUDE) -o $(TARGET4) rvitest_main.o $(LIBS)
//
/**
 * Constante représentant une chaîne de caractères ('clean' du 'Makefile')
 */
String clean_and_depend = "clean:\n\trm $(TARGETS) *.o\ninclude Makefile.deps\n";

////////////////////////////////////////////////////////////////////////////
/// Constructor
/**
 * Constructeur principal de la classe.
 * @param filename (fichier de sortie)
 * @param path     (chemin du répertoire de sortie)
 * @param plato_path (chemin vers l'application PLATO)
 */
public Makefile(String filename, String path, String plato_path) {
    this.filename = filename;
    this.path = path;
    this.plato_path = plato_path;
}
////////////////////////////////////////////////////////////////////////////
/// Functions
/**
 * Fonction ajoutant un fichier cible au 'Makefile'.
 * @param filename
 * @return
 */
public void addTarget(String filename) {
    targets_filename.add(filename);
}
/**
 * Fonction retournant la chaîne de caractères représentant le 'Makefile'.
 * @return
 */
public String getOutput() {
    StringBuilder output = new StringBuilder();   //--Output content
    //--Add the header of the makefile
    output.append(start);
    //--Add the target and create and Add the new target String
    //--Note: the target is i+2 since TARGET1 is ../lib/libplato.a
    String t = "TARGETS = $(TARGET1) ";
    for (int i = 0; i < targets_filename.size(); i++) {
        output.append("TARGET" + (i + 2) + " = " + targets_filename.get(i) + "\n");
        t += "$(TARGET" + (i + 2) + ") ";
    }
    t += "\n";
    output.append(t);
    output.append(o);
    output.append(all);
    //--For each target, add the build instruction
    for (int i = 0; i < targets_filename.size(); i++) {
        output.append("$(TARGET" + (i + 2) + ") : $(TARGET1) "
        + targets_filename.get(i)
        + ".o\n\t" + "$(CC) $(CFLAGS) -I$(INCLUDE) -o $(TARGET" + (i + 2) + ") "
        + targets_filename.get(i) + ".o $(LIBS)\n");
    }

    //--Add final String
    output.append(clean_and_depend);
    return output.toString();
}
```

```
    /**
     * Fonction retournant le contenu du fichier de commandes (run.sh) menant
     * à l'exécution du 'Makefile'.
     * @return
     */
    public String getCommand() {
        String command = "";
        //--Make
        //--remove previous usagers/xxx plato directory
        command += "rm -rf " + path + "plato.1.1\n";
        //--copy a fresh plato.1.1 directory structure to usagers/xxx
        command += "cp -r " + plato_path + " " + path + "\n";
        //--copy the Makefile
        command += "cp " + filename + " " + path + "plato.1.1/src/Makefile\n";
        //--copy the *.cc to the plato.1.1/src directory
        command += "cp " + path + "*.cc " + path + "plato.1.1/src\n";
        //--Go into this directory and make the objects
        command += "cd " + path + "plato.1.1/src/\n";
        //command+="make clean\n";
        command += "make all\n";
        for (int i = 0; i < targets_filename.size(); i++) {
            command += "chmod +x " + targets_filename.get(i) + "\n";
        }

        return command;
    }
}
```

Tableau E.11 - Le code source de la classe *Makefile.java*.

Le code source de la classe *StructureBoucleCetus.java* est présenté dans le Tableau E.12.

```
/* La classe StructureBoucleCetus.java */
package bouclesablecc;

import java.util.ArrayList;

/**
 * Classe implémentant une structure contenant les informations sur les boucles
 * for traitées lors de l'analyse des dépendances des données par l'application Cetus.
 * @author Alpha Boubacar Diallo et Etienne Lord
 * @since 2012
 */
public class StructureBoucleCetus {
    ////////////////////////////////////////////////////////////////////////////
    /// VARIABLES
    /**
     * Variable identifiant une boucle 'for' parallèle détectée par Cetus.
     */
    String loop_private = "";
    /**
     * Variable identifiant le nom de la boucle 'for' donné par Cetus.
     */
    String loop_name = "";
    /**
     * Variable contenant une représentation de la boucle 'for' par Cetus.
     */
    String loop_for = "";
    /**
     * Variable contenant la ou les dépendances trouvées par Cetus.
     */
    String loop_dependence = "";
    /**
     * Variable indiquant la présence ou l'absence de dépendance trouvées par Cetus.
     */
    boolean dependence_found = false;
    /**
     * Variable contenant les dépendances de type 'scalar' détectées par Cetus.
     */
    String scalar_dependence = "";
    /**
     * Variable contenant les dépendances de type 'array' détectées par Cetus.
     */
    String array_dependence = "";
    /**
     * Variable indiquant une boucle 'for' de type 'serial' pour l'application Cetus.
     */
    boolean serial = false;
    /**
     * Variable indiquant une boucle 'for' de type 'parallel' pour l'application Cetus.
     */
    boolean parallel = false;
    /**
     * Variable temporaire indiquant l'imbrication de la boucle 'for'.
     * Utilisé lors de l'affichage sur la page web.
     */
    int level = 0;
    /**
     * Variable temporaire (2) indiquant l'imbrication de la boucle 'for'.
     * Utilisé lors de l'affichage sur la page web.
     */
    int indentlevel = 0;
    /**
     * Liste contenant les boucles 'for' imbriquées dans cette boucle 'for'
     * considéré comme le parent.
     */
    ArrayList<StructureBoucleCetus> childs = new ArrayList<StructureBoucleCetus>();


    ////////////////////////////////////////////////////////////////////////////
    /// CONSTRUCTOR
    /**
```

```
 * Constructeur principal de la classe.
 */
public StructureBoucleCetus() {
}

////////////////////////////////////////////////////////////////////////////
/// FUNCTIONS
/**
 * Fonction retournant une représentation de boucle 'for' détectée par Cetus.
 * @return Une représentation de cette structure.
 */
@Override
public String toString() {
    if (childs.isEmpty()) {
        return loop_name + "\n" + loop_for;
    } else {
        String tmp = loop_name + "\n\t";
        for (StructureBoucleCetus e : childs) {
            tmp += e.toString() + "\n";
        }
        return tmp;
    }
}

/**
 * Fonction vérifiant si une boucle 'for' est un enfant de ce parent.
 * @param e
 * @return true si la boucle spécifiée est un enfant de cette boucle 'for'.
 */
boolean isChild(StructureBoucleCetus e) {
    return (e.loop_name.startsWith(loop_name));
}

/**
 * Fonction insérant une boucle 'for' comme un enfant de cette boucle (parent).
 * @param e
 * @return true en cas de succès.
 */
boolean insertChild(StructureBoucleCetus e) {
    if (!isChild(e)) {
        return false;
    }
    if (childs.isEmpty()) {
        childs.add(e);
        return true;
    } else {
        for (StructureBoucleCetus a : childs) {
            if (a.isChild(e)) {
                return a.insertChild(e);
            }
        }
        //--Not inserted?
        childs.add(e);
        return true;
    }
} //--End insert child

/**
 * Fonction retournant les enfants de cette boucle 'for'.
 * @param e
 * @return Les enfants de cette boucle 'for'.
 */
StructureBoucleCetus returnChild(StructureBoucleCetus e) {
    if (childs.isEmpty()) {
        return null;
    }


    //--Check imediate
    for (StructureBoucleCetus a : childs) {
        if (e.loop_name.equals(a.loop_name)) {
            return a;
        }
    }
```

```
        //--Check childs
        for (StructureBoucleCetus a : childs) {
            //--Child children
            if (a.isChild(e)) {
                return a.returnChild(e);
            }
        }
        return null;
    }

    /**
     * Fonction enlevant un enfant de cette boucle 'for'.
     * @param e
     * @return l'enfant enlevé.
     */
    StructureBoucleCetus removeChild(StructureBoucleCetus e) {
        if (childs.isEmpty()) {
            return null;
        }
        //--Check imediate
        for (int i = 0; i < childs.size(); i++) {
            StructureBoucleCetus a = childs.get(i);
            if (e.loop_name.equals(a.loop_name)) {
                childs.remove(i);
                return a;
            }
        }
        //--Check childs
        for (StructureBoucleCetus a : childs) {
            //--Child children
            if (a.isChild(e)) {
                return a.removeChild(e);
            }
        }
        return null;
    }
}
```

Tableau E.12 - Le code source de la classe *StructureBoucleCetus.java*.

Le code source de la classe *StructureBouclePlato.java* est présenté dans le Tableau E.13.

```
/* La classe StructureBouclePlato.java */
package bouclesablecc;
import java.util.Vector;

/**
 * Classe implémentant une structure contenant les informations sur les boucles
 * 'for' traitées lors de l'analyse des dépendances des données par l'application PLATO.
 * @author Alpha Boubacar Diallo
 * @since June 2011
 */
public class StructureBouclePlato {

        ////////////////////////////////////////////////////////////////////////
        /// VARIABLES

       /**
         * Liste des variables recevant une valeur.
         */
        public  Vector<String> inputAssignement;
        /**
         * Liste des variables donnant une valeur.
         */
        public  Vector<String> outputAssignement;
        /**
         * Variable des indices.
         */
        public  String inputIndices;
        /**
         * Variable contenant les représentations en chaînes de caractères des boucles
         *'for'.
         */
        public  Vector<String> inputBouclesFor;
        /**
         * Variable contenant toutes les boucles 'for'.
         */
        public String boucleFor;
        /**
         * Variable contenant l'indice maximal des boucles 'for'.
         */
        public int maxIndice=1;

        ////////////////////////////////////////////////////////////////////////
        /// CONSTRUCTOR

        /**
         * Constructeur principal de cette structure.
         */
        StructureBouclePlato(){
            this.inputAssignement   =new Vector<String>();
            this.outputAssignement  =new Vector<String>();
            this.inputIndices       =new String();
            this.boucleFor          =new String();
            this.inputBouclesFor    =new Vector<String>();
            this.maxIndice=1;
        }

}
```

Tableau E.13 - Le code source de la classe *StructureBouclePlato.java*.

Le code source de la classe *VariableData.java* est présenté dans le Tableau E.14.

```
/* La classe VariableData.java */
package bouclesablecc;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.util.HashMap;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

/**
 * Classe servant à l'identification des différents composants des structures  'struct'.
 * Ces  informations serviront à la génération des structures MPI utilisées lors
 * de la détection des dépendances par Cetus.
 * @author Etienne Lord
 * @since February 2012
 */
public class VariableData {

    ////////////////////////////////////////////////////////////////////////////
    /// Constant
    /**
     * Variable servant à indiquer le mode déboguage de la classe.
     */
    boolean debug = true;
    ////////////////////////////////////////////////////////////////////////////
    /// Datatype and variables
    /**
     * Type de données interne propre à cette classe.
     */

    public class Data {
        String name = "";
        String type = "";
        String mpi_type = "";
        boolean array = false;
        boolean constant = false;
        boolean pointer = false;
        String value = "";
        int array_size = 0;
        int array_level = 0;

        @Override
        public String toString() {
            return (type + " " + name + " (array? " + array + ")");
        }
    }
    /**
     * Liste des types de données trouvés dans la structure.
     * La clé est le nom de la variable et la valeur est une structure contenant
     * des informations e.g. type, nombre, array...
     */
    //--This is the extracted datafile...
    public HashMap<String, Data> dataArray = new HashMap<String, Data>();
    /**
     * Liste des '#define' trouvé dans le fichier source.
     */
    private HashMap<String, String> defineArray = new HashMap<String, String>();//define#
    /**
     * Variable temporaire contenant le fichier source à traiter.
     */
    //--This is the complete input file
    StringBuilder input_string_buffer = new StringBuilder();
    ////////////////////////////////////////////////////////////////////////////
    /// Pattern matching
    private Pattern struc_array;//--REGEX: Pattern to find int e.g.a[...]

    //REGEX: Pattern to have the inside of the a[xxx] -> xxx
    private Pattern struc_array_inside;
    private Pattern struc_comment;//--REGEX: Patern for the struct
    //comment find in file
```

```
    private Pattern valid_type;  //--REGEX: Pattern for the line
    // (ex. C++) e.g.  l; or long l=1;
    private Pattern equal_type;  //--REGEX: Pattern for the line
    //(ex. C++) e.g.  l; or long l=1;
    private Pattern complex_type;//--REGEX: Pattern for the line
    //(ex. C++) e.g. int c, d, e; or int c=1, d, e;
    private Pattern struc_define;//--REGEX: Pattern to find #define

    ////////////////////////////////////////////////////////////////////////////
    /// Constructor
    /**
     * Constructeur principal de la classe utilisée par Autoserial.
     */
    public VariableData() {
        //--Initialize some search pattern
        //--Pattern to match e.g. void (...); (single line)
        struc_comment = Pattern.compile("(.*?)[(].*?[)].*?;");
        //--Pattern to match e.g. int a[...]; or a[...][...]...; (single line)
        struc_array = Pattern.compile("(.*?)[\\[](.*?)[\\]](.*?;)(.*)");
        //--Pattern to match e.g. [xxx] -> (single line)
        struc_array_inside = Pattern.compile("[\\[](.*?)[\\]]");
        //--Pattern to match e.g. #define (single line)
        struc_define = Pattern.compile("#define\\s+(.*?)\\s+(.*)", Pattern.DOTALL);
        valid_type = Pattern.compile("(char|int|double|float|long|short)\\s+(.*?);");
        complex_type = Pattern.compile("(.*?)\\s+(.*?);");
        equal_type = Pattern.compile("(.*)=");
    }

    /**
     * Fonction essayant de faire une correspondance entre la 'variable' et la
     * boucle 'for' en entrée.
     * Note: si la variable n'est pas trouvée, elle est ajouté à la liste 'dataArray'.
     * Note: This should be called after process_file
     * Note: This should have been really done in SableCC
     * @param for_loop
     */
    public void process_for_loop(String for_loop, String variable) {
        if (dataArray.containsKey(variable)) {
            //System.out.println(variable+"->"+dataArray.get(variable).toString());
        } else {
            //--Scan for variable and add
            int indexArray = for_loop.indexOf(variable + " [");
            int indexAssign = for_loop.indexOf(variable + " =");
            // 1. We have an array
            if (indexArray > -1) {
                Data d = new Data();
                d.name = variable;
                d.array = true;
                dataArray.put(variable, d);
            } else if (indexAssign > -1) {
                Data d = new Data();
                d.name = variable;
                d.array = false;
                dataArray.put(variable, d);
            }
        }
    }
    /**
     * Fonction principale de la classe permettant de détecter les structures
     * 'struct' et de générer une représentation compatible pour Cetus.
     * @param infile
     * @return
     */
    public boolean process_file(String infile) {
        //1. Able to read file?
        if (!readfile(infile)) {
            return false;
        }
        //2. look for type
        //2.1 First add the define
        for (String s : defineArray.keySet()) {
            Data d = new Data();
            d.name = s;
            d.value = defineArray.get(s);
```

```
            d.constant = true;
            d.mpi_type = "";
            dataArray.put(s, d);
        }
        //2.2 Scan for type (note: like in transform-mpi);
        StringBuilder st = new StringBuilder();
        for (String str : this.input_string_buffer.toString().split("\n")) {
            //--1. Handle line like:
            // char    i,j,k;
            //--By first spliting the line into 3 char i; char j; char k;
            String[] substri = str.trim().split(",");
            //--Found possible multiple declaration on one line
            if (substri.length > 1) {
                //--Is it enum or comment?
                if (struc_comment.matcher(str).find() || str.contains("ENUM")
                        || str.contains("enum") || str.trim().startsWith("//")) {
                    //--Do nothing - Really just a normal line... -> create a one line
                    // buffer
                    substri = new String[0];
                    substri[0] = str;
                } else {
                    //--Ok, the real deal, split the definition
                    //1. What we have?
                    //--Add ; to the end of substri[0]
                    substri[0] = substri[0] + ";";
                    Matcher m_type = complex_type.matcher(substri[0]);
                    if (m_type.find()) {
                        String ntype = m_type.group(1);
                        for (int i = 1; i < substri.length; i++) {
                            substri[i] = ntype + " " + substri[i].trim();
                            if (!substri[i].endsWith(";")) {
                                substri[i] = substri[i] + ";";
                            }
                        }
                    }
                }
            } //--End substri>1

            //--Find the type and name
            String type = "";
            String data = "";    //--or name
            for (String stri : substri) {
                //System.out.println("AutoSerial:"+stri);
                Matcher m_valid = valid_type.matcher(stri);
                if (m_valid.find()) {
                    type = m_valid.group(1);
                    data = m_valid.group(2);
                } else {
                    Matcher m_complex = complex_type.matcher(stri);
                    if (m_complex.find()) {
                        type = m_complex.group(1);
                        data = m_complex.group(2);
                    }
                }
                //--Final clean up
                //--Remove: =
                if (data.indexOf("=") > -1) {
                    data = data.substring(0, data.indexOf("="));
                }

                //--Break if data or type is null
                if (data.isEmpty() || type.isEmpty()) {
                    //--Empty data
                } else {
                    Matcher m_struc_array = struc_array.matcher(stri);
                    //--CASE 1. Array?
                    if (m_struc_array.find()) {
                        //--Determine the size
                        Matcher m_stru_array_inside = struc_array_inside.matcher(stri);
                        int length = 1;
                        int level = 0;
                        //--Flag to know if we removed some information in the
                        // structures...
                        boolean flag_removed = false;
```

```
                    while (m_stru_array_inside.find()) {
                        String inside = m_stru_array_inside.group(1);
                        level++;
                        //--If it is a number... keep it
                        try {
                            int i = Integer.valueOf(inside);
                            length *= i;
                        } catch (Exception e) {
                            // Else... replace it
                            flag_removed = true;
                        }
                    } //--End while
                    //--Keep the name til [ bracket
                    Data d = new Data();
                    data = data.substring(0, data.indexOf('['));
                    d.array = true;
                    d.name = data;
                    d.array_size = length;
                    d.array_level = level;
                    d.type = type;
                    d.mpi_type = getMPI_type(type);
                    dataArray.put(data, d);
                } //--CASE 2. No array
                else {
                    if (debug) {
                        System.out.println(data + ":" + type);
                    }
                    //--Clean up
                    if (!type.startsWith("for") && !type.startsWith("return")) {
                        //--Pointer?
                        if (data.startsWith("*")) {
                            Data d = new Data();
                            d.name = data.substring(1);
                            d.pointer = true;
                            d.type = type;
                            d.array = true;
                            d.array_size = 0;
                            d.mpi_type = getMPI_type(type);
                            dataArray.put(d.name, d);
                        } else {
                            Data d = new Data();
                            d.name = data;
                            d.pointer = false;
                            d.type = type;
                            d.array = false;
                            d.array_size = 0;
                            d.mpi_type = getMPI_type(type);
                            dataArray.put(d.name, d);
                        }
                    }
                }
            } //--End no data
        } //--End for stri
    } //--End for str
    if (debug) {
        System.out.println("===Results");
        for (Data d : dataArray.values()) {
            System.out.println(d);
        }
    }
    return true;
}

/**
 * Fonction retournant un 'MPI_type' valid correspondant à un 'type' en C.
 * @param type
 * @return Le type MPI ou rien si non trouvé.
 */
private String getMPI_type(String type) {
    if (type.equals("char")) {
        return "MPI_CHAR";
    }
    if (type.equals("signed char")) {
        return "MPI_SIGNED_CHAR";
```

```
        }
        if (type.equals("unsigned char")) {
            return "MPI_UNSIGNED_CHAR";
        }
        if (type.equals("short")) {
            return "MPI_SHORT";
        }
        if (type.equals("unsigned short")) {
            return "MPI_UNSIGNED_SHORT";
        }
        if (type.equals("signed int") || type.equals("int")) {
            return "MPI_INT";
        }
        if (type.equals("unsigned int")) {
            return "MPI_UNSIGNED";
        }
        if (type.equals("signed long") || type.equals("long")) {
            return "MPI_LONG";
        }
        if (type.equals("unsigned long")) {
            return "MPI_UNSIGNED_LONG";
        }
        if (type.equals("float")) {
            return "MPI_FLOAT";
        }
        if (type.equals("double")) {
            return "MPI_DOUBLE";
        }
        if (type.equals("byte")) {
            return "MPI_BYTE";
        }
        return ""; //--Not found
    }

    /**
     * Fonction lisant et réalisant un premier nettoyage du fichier en entrée
     * @param infile (in file)
     * @return true if success, false otherwise
     */
    private boolean readfile(String infile) {
        try {
            //--Open the file for reading
            BufferedReader br = new BufferedReader(new FileReader(new File(infile)));
            String buffer = "";             //--Buffer for escape \ line
            System.out.println("===Pre-Processing input file (" + infile + "):");
            //--Read the file line by line
            while (br.ready()) {
                // --Process the string to remove some error
                String stri = br.readLine();

                //--1.1 handle \ case
                if (stri.trim().isEmpty() && !buffer.isEmpty()) {
                    stri = buffer;
                    buffer = "";
                    Matcher m1 = struc_define.matcher(stri);
                    if (m1.find()) {
                        String key = m1.group(1);
                        String value = m1.group(2);
                        //--We only keep valid define key for types
                        if (key.indexOf("(") == -1
                                && key.indexOf(")") == -1
                                && key.indexOf(":") == -1) {
                            defineArray.put(key, value);
                        }
                    } //--End if m1.find()
                } //--End case 1.1

                //--1. Remove any line comments: // and single \
                int index = stri.indexOf("//");
                int index_single = stri.indexOf("\\");
                //int index_comment=stri.indexOf("/**");
                //int index_comment_out=stri.indexOf("**/");
                //--case Double bracket //
                if (index != -1) {
```

```
                    stri = stri.substring(0, index) + "/*" + stri.substring(index + 2) +
                    "*/";
                    if (debug) {
                        System.out.println("Warning (change of comments): " + stri);
                    }
                }
                //-- case  escape \
                if (index_single != -1) {
                    System.out.println("Warning (change of single comments): " + stri);
                    stri = stri.replace("\\", " ");
                    buffer += stri + "\n";
                }
                //--Finally, append to the input buffer
                input_string_buffer.append(stri + "\n");
            }
            br.close();
            System.out.println("===Definition found(s):");
            //--Store in memory some #define found for further use in the transformation
            for (String s : defineArray.keySet()) {
                System.out.println(s + ": " + defineArray.get(s));
            }
            System.out.println("===Done PreProcessing");
            return true;
        } catch (Exception e) {
            return false;
        }
    } //--End readfile
}
```

Tableau E.14 - Le code source de la classe *VariableData.java*.

Le code source de la classe *WebPage.java* est présenté dans le Tableau E.15.

```
/* La classe WebPage.java */

package bouclesablecc;

import bouclesablecc.VariableData.Data;
import java.io.File;
import java.io.FileWriter;
import java.io.PrintWriter;
import java.io.PushbackReader;
import java.io.StringReader;
import java.net.URLEncoder;
import java.util.ArrayList;
import java.util.Collections;
import java.util.HashMap;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

/**
 * Classe servant à générer les pages web servant de sortie sur le site web
 * 'transform to MPI'.
 * @author Alpha Boubacar Diallo et Etienne Lord
 * @since July 2011
 */
public class WebPage {

    ////////////////////////////////////////////////////////////////////////////
    /// Class to keep the loop structure (in variable, out variable)
    /**
     * Structure interne de données.
     */
    public class loop_struct {
        String loop;
        String synchronisation_variable;        // Indice used to devide the loop
        ArrayList<String> out;                  // Input variables
        ArrayList<String> in;                   // Output variables
        public ArrayList<String> antidep;   //--Antidependence variables found.
        public Boolean antidependence = false; //--Antidependence found.

        public loop_struct() {
            clear();
        }

        @Override
        public loop_struct clone() {
            loop_struct r = new loop_struct();
            r.loop = this.loop;
            r.synchronisation_variable = this.synchronisation_variable;
            for (String s : this.in) {
                r.in.add(s);
            }
            for (String s : this.out) {
                r.out.add(s);
            }
            for (String s : this.antidep) {
                r.antidep.add(s);
            }
            r.antidependence = this.antidependence;
            return r;
        }

        public void clear() {
            loop = "";
            out = new ArrayList<String>();
            in = new ArrayList<String>();
            antidep = new ArrayList<String>();
            antidependence = false;
        }

        /**
         * Add an in variable found in the loop
         * -- Need synchronisation --
```

```
     * @param s
     */
    public void addIn(String s) {
        if (!in.contains(s) && !isNumeric(s)) {
            in.add(s);
        }
    }

    /**
     * Add an out variable found in the loop
     * -- The out variable is the variable that will need to decompose --
     * @param s
     */
    public void addOut(String s) {
        //--We test to that we don't have the variable
        //already and the variable is not numeric
        if (!out.contains(s) && !isNumeric(s)) {
            out.add(s);
        }
    }

    /**
     * Add the loop description to the structure
     * @param s
     */
    public void addLoop(String s) {
        this.loop = s;
    }

    /**
     * Add the loop description to the structure
     * @param s
     */
    public void addSync(String s) {
        this.synchronisation_variable = s;
    }

    /**
     * Add the antidependence variables  to the structure
     * @param s
     */
    public void addAntidependence(String s) {
        this.antidep.add(s);
    }

    /**
     * Test if the variable is numeric (e.g. 10, 20, 1...)
     * @param s
     * @return  true if numeric variable, false otherwise
     */
    public boolean isNumeric(String s) {
        try {
            Integer i = Integer.valueOf(s);
            return true;
        } catch (Exception e) {
            return false;
        }
    }
}
////////////////////////////////////////////////////////////////////////////
/// Variables
/**
 * Liste contenant les résultats des tests sur chacunes des boucles 'for'.
 */
// Array for the tests results
public static ArrayList<String> results = new ArrayList<String>();


/**
 * Liste contenant les boucles 'for' associés aux résultats.
 */
// Array for the tests results (loop)
public static ArrayList<String> loop = new ArrayList<String>();
/**
```

```
 * Liste contenant les structures de boucle 'for' à ajouter à la page web.
 */
public static ArrayList<loop_struct> loop_transform = new ArrayList<loop_struct>();
/**
 * Variable temporaire contenant une structure de boucle 'for'.
 */
public loop_struct current_loop_struct = new loop_struct();
/**
 * Variable contenant le chemin de sortie de la page web générée.
 */
public static String path = "";

////////////////////////////////////////////////////////////////////////////
/// Constructor
/**
 * Default constructon
 * @param path (path if the output path is not the same directory
 */
public WebPage(String path) {
    this.path = path;
}

////////////////////////////////////////////////////////////////////////////
/// Main generator function
/**
 * Main function to generate the web page.
 * Note: Also output informations in System.out
 * @param filename
 * @return true if success, false otherwise
 */
public boolean generate(String filename) {
    try {
        ////////////////////////////////////////////////////////////////////
        /// REGEX Patterns for
        /// VI-Test proved dependence for system
        /// VI-Test proved dependence for equation
        /// Dependence proved

        String independence_str = "VI-Test proved independence for system";

        PrintWriter pw = new PrintWriter(new FileWriter(new File(filename)));

        for (int i = 0; i < this.results.size(); i++) {
            //--Test if we have a results... we never know...


            //--Trace
            System.out.println("");
            String stars = "**********************************************";
            System.out.println(stars);
            String s = String.format("*             Results for loop %d*", i + 1);
            System.out.println(s);
            System.out.println(stars);
            System.out.println(parse(this.loop.get(i)));
            String result_str = this.results.get(i);
            String[] array_str = result_str.split("\n");
            //System.out.println ("\n result = " + result_str);

            String loop_str = stars + "\nLoop" + stars + "\n"
                    + parse(WebPage.loop.get(i));
            String formated_results = stars + "\nPlato 1.1 trace" + stars + "\n"
                    + result_str + stars;
            if (!OutputFile(this.path + "loop_dependency_trace" + (i + 1) + ".txt",
                    loop_str + formated_results)) {
                System.out.println("Warning, unable to output "
                        + "loop_dependency_trace" + (i + 1) + ".txt");
            }
            if (result_str.startsWith("Plato error")) {
                pw.println("<div class='ui-widget'><div class='ui-state-error "
                        + "ui-corner-all' style='padding: 0 .7em;'><p>"
                        + "<span class='ui-icon ui-icon-alert' "
                        + "style='float: left; margin-right: .3em;'></span>");
                pw.println("<strong>Warning. Plato v1.1 was unable to perform "
                        + "the dependence test for this loop.</strong><br><br>");
```

```
            }
            else if (independence_str.equalsIgnoreCase(array_str[array_str.length -
            1])) {
                //--CASE 1. Ok, no dependence found.
                pw.println("<div class='ui-widget'><div class='ui-state-highlight "
                        + "ui-corner-all' style='margin-top: 20px; padding: 0" .
                        +"7em;'>"
                        + "<p><span class='ui-icon ui-icon-info' style='float: "
                        + "left; margin-right: .3em;'></span>");
                pw.println("<strong>No Flow dependency found.</strong><br><br>");
                System.out.println("Dependence not proved");
            } else {
                //--CASE 2. Found some dependence issue
                pw.println("<div class='ui-widget'><div class='ui-state-error"
                + "ui-corner-all' style='padding: 0 .7em;'><p><span class='ui-icon"
                +"ui-icon-alert' "
                + "style='float: left; margin-right: .3em;'></span>");
                pw.println("<strong>Warning. Flow Dependency found. "
                +"</strong><br><br>");
            }

            //--For output (HTML)
            pw.println(this.parseHTML_Transformation(this.loop.get(i)) + "<br>\n");

            //--HTML trace
            pw.println("<br><a href='" + this.path + "loop_dependency_trace" +(i + 1)
                    + ".txt' target='_blank'>computation details</a><br>");
            //--Close the jQuery statement

            //Add antidependencies part : bouba 17-Dec-2011
            try {
                if (this.loop_transform.get(i).antidependence) {
                    pw.println("<div class='ui-widget'><div class='ui-state-error"
                   +"ui-corner-all' style='padding: 0 .7em;'><p><span class='ui-icon"
                   +"ui-icon-alert' style='float: left; margin-right: ."
                   +"3em;'></span></p>");
                    pw.println("<strong>Warning. Antidependency found.<br>");
                    if (this.loop_transform.get(i).antidep.size() == 1) {
                        pw.print("     Variable is : ");
                    } else {
                        pw.print("     Variable are : ");
                    }
                    for (int k = 0; k < this.loop_transform.get(i).antidep.size();
                    k++) {
                        pw.print(this.loop_transform.get(i).antidep.get(k));
                        if (k == this.loop_transform.get(i).antidep.size() - 1) {
                            pw.print(".");
                        } else {
                            pw.print(",");
                        }
                    }
                    pw.println("</strong><br><br></div><br></div>");
                }
            } catch (Exception e2) {
            }

            pw.println("</div></div><br>");
            System.out.println("");
        } //--End for

        pw.flush();
        pw.close();
        //--TO DO Create a README file for the user directory...
    } catch (Exception e) {
        e.printStackTrace();
        return false;
    }
    return true;
}

/**
 * Main function to generate the web page for Cetus
 * Note: Also output informations in System.out
 * @param filename
```

```
     * @return true if success, false otherwise
     */
    public boolean generateCetus(String filename, ArrayList<StructureBoucleCetus> boucle)
    {try {
            PrintWriter pw = new PrintWriter(new FileWriter(new File(filename)));
            int last_level = 1;
            int total_level = 0;
            pw.println("<div class='ui-widget'>");
            for (int i = 0; i < boucle.size(); i++) {
                //--Test if we have a results... we never know...
                //--Trace
                System.out.println("");
                String stars = "*********************************************";
                System.out.println(stars);
                String s = String.format("*             Results for loop %d *", i + 1);
                System.out.println(s);
                System.out.println(stars);
                System.out.println(parse(boucle.get(i).loop_for));
                String loop_str = stars + "\nLoop" + stars + "\n"
                + parse(boucle.get(i).loop_for);
                if (last_level >= boucle.get(i).level) {
                    for (int j = boucle.get(i).level; j <= last_level; j++) {
                        pw.println("</div>");
                        total_level--;
                    }
                    pw.println("<br>");
                }
                last_level = boucle.get(i).level;
                total_level++;
                if (!boucle.get(i).dependence_found) {
                    //--CASE 1. Ok, no dependence found.
                    pw.println("<div class='ui-state-highlight ui-corner-all' "
                            + "style='margin-top: 20px; padding: 0 .7em;'><p>");
                    pw.println("<span class='ui-icon ui-icon-info' style='float: "
                            + "left; margin-right: .3em;'></span><strong>No "
                            + "dependency found.</strong><br>");
                    System.out.println("Dependence not proved");
                } else {
                    //--CASE 2. Found some dependence issue
                    pw.println("<div class='ui-state-error ui-corner-all' "
                            + "style='padding: 0 .7em;'><p>");
                    if (!boucle.get(i).scalar_dependence.isEmpty()) {
                        pw.println("<span class='ui-icon ui-icon-alert' style='float: "
                                + "left; margin-right: .3em; '></span><strong>Scalar "
                                + "dependences</strong> detected on: <b>"
                                + boucle.get(i).scalar_dependence + "</b><br>");
                        System.out.println("Scalar dependences detected on: "
                                + boucle.get(i).scalar_dependence);
                    }
                    if (!boucle.get(i).array_dependence.isEmpty()) {
                        pw.println("<span class='ui-icon ui-icon-alert' style='float:"
                                + "left; margin-right: .3em;'></span><strong>Array "
                                + "dependences</strong> detected on: <b>"
                                + boucle.get(i).array_dependence + "</b><br>");
                        System.out.println("Array dependences detected on: "
                                + boucle.get(i).array_dependence);
                    }
                }

                //--For output (HTML)
                pw.println(this.parseHTML_Transformation(boucle.get(i).loop_for) + "\n");
            } //--End for
            if (last_level > 1) {
                for (int j = 1; j <= last_level; j++) {
                    pw.println("</div>");
                    total_level--;
                }
            }
            if (total_level == 0) {
                pw.println("</div>");
            }
            pw.println("<br>");
            //--Trace for all loop:
            pw.println("Note:<br><span class='ui-icon ui-icon-bullet' style='float:"
```

```
                + " left; margin-right: .3em;'></span>Indentation level are "
                + "indicative of inner-loop levels as detected by Cetus. <br>"
                + "<span class='ui-icon ui-icon-bullet' style='float: left; "
                + "margin-right: .3em;'></span>Also note that Cetus introduce a "
                + "\"<b>return</b>\" statement at the end of the last loop as "
                + "part of its dependency detection procedure.<br>");
        pw.println("<br><a href='" + this.path + "loop_dependency_trace.txt' "
                + "target='_blank'>computation details</a><br>");
        pw.flush();
        pw.close();
        //--TO DO Create a README file for the user directory...

    } catch (Exception e) {
        e.printStackTrace();
        return false;
    }
    return true;
}

/**
 * Main function to generate the web page.
 * Note: Also output informations in System.out
 * @param filename
 * @param data (processed file for data type, array...)
 * @return true if success, false otherwise
 */
public boolean generateLoop(String filename, VariableData data) {

    try {
        PrintWriter pw = new PrintWriter(new FileWriter(new File(filename)));
        //--jQuery for accordion
        pw.println("<script>$(function() {     $( '#accordion' ).accordion"
                + "({collapsible: true});});</script>");
        pw.println(" <div id='accordion'>");
        //--End jQuery
        for (int i = 0; i < loop_transform.size(); i++) {
            //--VAriables for the print=out of the variables to synchronize
            String variables_html = ""; //--HTML print out
            String variables_txt = "";  //--TXT print out
            loop_struct l = loop_transform.get(i);
            System.out.println("\nLoop " + (i + 1) + ":");
            pw.println("<h3><a href='#" + (i + 1) + "'>Loop " + (i + 1)
            + "</a></h3><div><p>");
            pw.println("<div class='ui-widget'><div class='ui-state-highlight "
                    + "ui-corner-all' style='margin-top: 20px; padding: 0"
                    + ". "7em;'><p>");
            pw.println("<table><tr><td><span class='ui-icon ui-icon-info' "
                    + "style='float: left; margin-right: .3em;'></span><strong>"
                    + "Suggested variable(s) to share before the loop beginning :"
                    + "</strong><br></td><td><table>");
            pw.println("<tr><td><b><span style=\"color: blue;\">");
            Collections.sort(l.out);

            if (l.out.size() > 0) {
                for (int y = 0; y < l.out.size(); y++) {
                    String s = l.out.get(y);
                    data.process_for_loop(l.loop, s);
                    if (y == l.out.size() - 1) {
                        variables_html += s + " ";
                        variables_txt += s + " ";
                    } else {
                        variables_html += s + ", ";
                        variables_txt += s + ", ";
                    }
                }
            } else {
                variables_html = "NONE";
            }
            System.out.println("Suggested variable(s) to share before the loop "
            +"beginning : " + variables_txt);
            pw.println(variables_html + "</span></b></td></tr>");
            pw.println("</tr></table></td></tr></table>");
            pw.println("</p></div></div><br>");
            System.out.println("Loop transformation: ");
```

```
//--Print here the distribution of Array to node
System.out.println("/*\n  Sample code distributing the data to nodes");
System.out.println("  Note: we assume the processorID 0 as "
        + "the master node\n */");
pw.println("/*<br>    Sample code distributing "
        + "the data to nodes<br>");
pw.println("    Note: we assume the master node."
        + "(processorID 0)<br> */<br>");
for (int y = 0; y < l.out.size(); y++) {
    String s = l.out.get(y);
    Data d = data.dataArray.get(s);
    //--Array
    if (data.dataArray.get(s).array) {
        String size = "";
        String type = d.mpi_type;
        if (type.isEmpty()) {
            type = "/* Put correct array type here  */";
        }
        if (d.array_size == 0) {
            size = "/* Put correct array size here */";
        } else {
            size = "" + d.array_size;
        }
        System.out.println("MPI_Bcast( " + d.name + ", " + size + ", "
                + type + ", 0, MPI_COMM_WORLD);");
        pw.println("MPI_Bcast( " + d.name + ", " + size + ", " + type
                + ", 0, MPI_COMM_WORLD);<br>");
    } else {
        String type = d.mpi_type;
        System.out.println("MPI_Bcast( " + d.name + ", 1, "
                + type + ", 0, MPI_COMM_WORLD);");
        pw.println("MPI_Bcast( " + d.name + ", 1, "
                + type + ", 0, MPI_COMM_WORLD);<br>");
    }
}
System.out.println("MPI_Barrier(MPI_COMM_WORLD);\n");
pw.println("MPI_Barrier(MPI_COMM_WORLD);<br>");
System.out.println("/*\n  Actual work divided for individual worker"
+"node\n*/");
pw.println("<br><b>/*<br>    Actual "
        + "work divided along individual worker node<br>*/</b><br>");
System.out.println(parse(InsertAfterKeyword("for", l.loop, "if ("
        + l.synchronisation_variable + " % nbProcessors == processorID)
        +"{")) + "\n");
pw.println("<br>" + parseHTML_Transformation(InsertAfterKeyword"
+"("for"," l.loop,"if (" + l.synchronisation_variable + " % nbProcessors"
+"== processorID) {")));
System.out.println("/*\n  Wait for each of the individual node to"
+"finish\n*/");
System.out.println("MPI_Barrier(MPI_COMM_WORLD);");
pw.println("<br>/*<br>    Wait for each of the "
        + "individual node to finish<br>*/<br>");
pw.println("MPI_Barrier(MPI_COMM_WORLD);<br><br>");
Collections.sort(l.in);
for (int y = 0; y < l.in.size(); y++) {
    String s = l.in.get(y);
    data.process_for_loop(l.loop, s);
}
System.out.println("/*\n  Retrieve data from each node \n*/");
System.out.println("if (processorID>0) {");
pw.println("/*<br>    Sample code (non-fonctionnal) "
        + "to <br>    Retrieve data from each node"
        + " to the master (processorID 0)<br>*/<br>");
pw.println("if (processorID>0) {<br>");

String loop1 = getFirstLoop("for", l.loop);
System.out.println("\t" + loop1 + "\n\t\tif ("
+l.synchronisation_variable+ " % nbProcessors == processorID) {");
pw.println("    " + loop1
+ "<br>        if ("
+ l.synchronisation_variable + " % nbProcessors == processorID) {<br>");

for (int y = 0; y < l.in.size(); y++) {
    String s = l.in.get(y);
```

```
                Data d = data.dataArray.get(s);
                System.out.println("\t\t\t/* --PSEUDO-CODE FOR EACH WORKERS "
                        + "SEND FOR THEIR PART OF " + s + "-- */");
                System.out.println("\t\t\tMPI_Send (Portion of " + s
                        + " processed by processorID with TAG processorID);");
                pw.println("        "
                + "    MPI_Send (<i>Portion of "
                + s + " processed by processorID with TAG processorID</i>);<br>");
            }
            System.out.println("\t\t}\n\t}");
            System.out.println("} else { ");
            System.out.println("\t" + loop1 + "");
            pw.println("        }"
                    + "<br>    }");
            pw.println("<br>} else { <br>");
            pw.println("    " + loop1 + "<br>");
            for (int y = 0; y < l.in.size(); y++) {
                String s = l.in.get(y);
                String loop = getFirstLoop("for", l.loop);
                Data d = data.dataArray.get(s);
                System.out.println("\t\t\t/* --PSEUDO-CODE FOR MASTER "
                        + "RECEIVED PART OF " + s + " FROM WORKERS" + "-- */");
                System.out.println("\t\t\tif (" + l.synchronisation_variable
                        + " % nbProcessors != 0) MPI_Recv (Portion of " + s
                        + " processed by processorID with TAG processorID);");
                pw.println("        "
                + "    MPI_Recv (<i>Portion of " + s
                + " processed by i % nbProcessor with TAG processorID</i>);<br>");
            }
            System.out.println("\t}\n}\n");
            pw.println("    }<br>}<br>");
            pw.println("<div class='ui-widget'><div class='ui-state-highlight "
            + "ui-corner-all' style='margin-top: 20px; padding: 0 .7em;'><p>");
            pw.println("<table><tr><td><span class='ui-icon ui-icon-info'"
            + " style='float: left; margin-right: .3em;'></span><strong>"
            + "Suggested variable(s) to synchronize after the loop ending:"
            + "</strong><br></td><td><table>");
            pw.println("<tr><td><b><span style=\"color: blue;\">");
            variables_html = "";
            variables_txt = "";
            if (l.in.size() > 0) {
                for (int y = 0; y < l.in.size(); y++) {
                    String s = l.in.get(y);

                    if (y == l.in.size() - 1) {
                        variables_html += s + " ";
                        variables_txt += s + " ";
                    } else {
                        variables_html += s + ", ";
                        variables_txt += s + ", ";
                    }
                }
            } else {
                variables_html = "NONE";
            }
            System.out.println("Suggested variable(s) to synchronize: "
                    + variables_txt);
            pw.println(variables_html + "</span></b></td></tr>");
            pw.println("</tr></table></td></tr></table>");
            pw.println("</p></div></div><br>");
            //System.out.println("Indice:\t"+l.synchronisation_variable);
            pw.println("</p></div>");
        }
        pw.println("</div>");
        pw.flush();
        pw.close();
    } catch (Exception e) {
        e.printStackTrace();
        return false;
    }
    return true;
}
```

```
////////////////////////////////////////////////////////////////////////////
/// Helper functions
/**
 * Parse a C code and generate a correct indented presentation string
 * The formatting is similar to the Whitesmiths style
 * The Whitesmiths style is also called Wishart style
 * See: http://en.wikipedia.org/wiki/Indent_style
 * @param str (C code)
 * @return a String with correct indentation
 */
public static String parse(String str) {
    String output = "";      //-- output buffer
    String last = "";        //-- Last read string
    int indent = 0;          //-- code indentation
    boolean flagFor = false; //--Flag to know if we found a "for" keyword
    boolean flagIf = false;  //--Flag to know if we found a "if" keyword
    boolean flagWhile = false;  //--Flag to know if we found a "if" keyword
    //--Flag to know if we found a for (...) without curly
    boolean flagSingleFor = false;
    int countFor = 0;          //--Counter for stats
    int countIf = 0;          //--Counter for stats
    int countWhile = 0;        //--Counter for stats
    int countParen = 0;
    //--Remove \n
    String s = "";
    for (char c : str.toCharArray()) {
        if (c != '\n') {
            s += c;
        }
    }
    for (int j = 0; j < s.length(); j++) {
        char c = s.charAt(j);
        switch (c) {
            case ' ':
                output += ' ';
                break;
            //case ' ': break;
            case '(':
                countParen++;
                if (last.replaceAll("\t", "").startsWith("for")) {
                    flagFor = true;
                    flagSingleFor = true;
                    last = "";
                    countFor++;
                }
                if (last.replaceAll("\t", "").startsWith("if")) {
                    flagIf = true;
                    last = "";
                    countIf++;
                }
                if (last.replaceAll("\t", "").startsWith("while")) {
                    flagWhile = true;
                    last = "";
                    countWhile++;
                }
                output += '(';
                break;
            case ')':
                countParen--;
                if ((flagFor || flagIf || flagWhile) && countParen == 0) {
                    flagFor = false;
                    flagIf = false;
                    flagWhile = false;
                    last = "";
                    indent++;
                    output += ")\n";
                    for (int i = 0; i < indent; i++) {
                        output += "    ";
                    }
                } else {
                    output += ")";
                }
                break;
            case ';':
```

```
                if (flagFor || flagIf || flagWhile) {
                    output += ";";
                } else {
                    output += ";\n";
                    last = "";
                    if (flagSingleFor) {
                        indent--;
                        flagSingleFor = false;
                    }
                    for (int i = 0; i < indent; i++) {
                        output += "    ";
                    }
                }
                break;
            case '{':
                flagSingleFor = false;
                flagFor = false;
                flagIf = false;
                flagWhile = false;
                last = "";
                //indent++;
                output += "{\n";
                for (int i = 0; i < indent; i++) {
                    output += "    ";
                }
                break;
            case '}':
                last = "";
                if (indent == 0) {
                    output += "    ";
                    output += "}\n";
                    indent--;
                } else {
                    indent--;
                    if (indent < 0) {
                        indent = 0;
                    }
                    output += "}\n";
                    for (int i = 0; i < indent; i++) {
                        output += "    ";
                    }
                }
                break;
            default:
                output += c;
                last += c;
        }
    }
    return output;
}

/**
 * Fonction encodant pour le web une chaîne de caractères.
 * @param string
 * @return une représentation UTF-8 de la chaîne de caractères.
 */
public static String encode(String string) {
    if (string == null || string.isEmpty()) {
        return "";
    }
    try {
        return URLEncoder.encode(string, "UTF-8");
    } catch (Exception e) {
        System.out.println(e.getMessage());
        return "";
    }
}

/**
 * Insert a directive after the first for-loop
 * @param str
 * @param toInsert (Note: must end with { )
 * @return The new string
 */
```

```
public static String InsertAfterKeyword(String keyword, String str, String toInsert) {

    //--Remove Blank for easier test
    String s = "";
    for (char c : str.toCharArray()) {
        if (c != ' ') {
            s += c;
        }
    }

    //--Find the first index of some constants
    int indexKeyword = s.indexOf(keyword);  //--Not used for now...
    int indexLCurly = s.indexOf("{", indexKeyword);
    //int indexRCurly=s.indexOf("}"); //--Not used for now...
    //int indexLParen=s.indexOf("("); //--Not used for now...
    int indexRParen = s.indexOf(")", indexKeyword);
    //--Case 1. for (...)
    if (indexLCurly != indexRParen + 1) {
        indexRParen = str.indexOf(")", str.indexOf(keyword));
        return str.substring(0, indexRParen + 1) + toInsert
                + str.substring(indexRParen + 1) + "}";
    }
    //--Case 2. for (...){...}
    if (indexLCurly == indexRParen + 1) {
        indexLCurly = str.indexOf("{", str.indexOf(keyword));
        return str.substring(0, indexLCurly + 1) + toInsert
                + str.substring(indexLCurly + 1) + "}";
    }
    return s;
}
/**
 * Insert a directive after the first for-loop
 * @param str
 * @param toInsert (Note: must end with { )
 * @return The new string
 */

public static String getFirstLoop(String keyword, String str) {
    //--Remove Blank for easier test
    String s = "";
    for (char c : str.toCharArray()) {
        if (c != ' ') {
            s += c;
        }
    }

    //--Find the first index of some constants
    int indexKeyword = s.indexOf(keyword);  //--Not used for now...

    int indexLCurly = s.indexOf("{", indexKeyword);
    //int indexRCurly=s.indexOf("}"); //--Not used for now...
    //int indexLParen=s.indexOf("("); //--Not used for now...
    int indexRParen = s.indexOf(")", indexKeyword);

    //--Case 1. for (...)
    if (indexLCurly != indexRParen + 1) {
        indexRParen = str.indexOf(")", str.indexOf(keyword));
        return str.substring(0, indexRParen + 1);
    }
    //--Case 2. for (...){...}
    if (indexLCurly == indexRParen + 1) {
        indexLCurly = str.indexOf("{", str.indexOf(keyword));
        return str.substring(0, indexLCurly + 1);
    }
    return "";
}

/**
 * Special parseHTML (generate HTML formated input for a C code)
 * The formatting is similar to the Whitesmiths style
 * The Whitesmiths style is also called Wishart style
 * See: http://en.wikipedia.org/wiki/Indent_style
 * This version of the function can add color to some variable
 *
```

```
     * @param str
     * @return a String containing valid HTML code
     */
    public static String parseHTML_Transformation(String str) {
        String output = "";      //-- output buffer
        String last = "";        //-- Last read string
        int indent = 0;          //-- code indentation
        boolean flagFor = false; //--Flag to know if we found a "for" keyword
        boolean flagIf = false;  //--Flag to know if we found a "if" keyword
        boolean flagWhile = false;  //--Flag to know if we found a "if" keyword

        //--Flag to know if we found a for (...) without curly
        boolean flagSingleFor = false;
        int countFor = 0;
        int countIf = 0;
        int countWhile = 0;
        int countParen = 0;
        //--Remove \n
        String s = "";
        for (char c : str.toCharArray()) {
            if (c != '\n') {
                s += c;
            }
        }

        for (int j = 0; j < s.length(); j++) {
            char c = s.charAt(j);
            switch (c) {
                case ' ':
                    output += " "; //TO DO, better parser for beauty...
                    break;
                case '(':
                    countParen++;
                    if (last.replaceAll("\t", "").startsWith("for")) {
                        flagFor = true;
                        flagSingleFor = true;
                        last = "";
                        countFor++;
                    }
                    if (last.replaceAll("\t", "").startsWith("if")) {
                        flagIf = true;
                        last = "";
                        countIf++;
                    }
                    if (last.replaceAll("\t", "").startsWith("while")) {
                        flagWhile = true;
                        last = "";
                        countWhile++;
                    }
                    output += " <b>(</b>";
                    break;
                case ')':
                    countParen--;
                    if ((flagFor || flagIf || flagWhile) && countParen == 0) {
                        flagFor = false;
                        flagIf = false;
                        flagWhile = false;
                        last = "";
                        indent++;
                        output += "<b>)</b><br>";
                        for (int i = 0; i < indent; i++) {
                            output += "    ";
                        }
                    } else {
                        output += "<b>)</b>";
                    }
                    break;
                case '<':
                    output += "&lt;";
                    break;
                case '>':
                    output += "&gt;";
                    break;
                case ';':
```

```
                    if (flagFor || flagIf || flagWhile) {
                        output += ";";
                    } else {
                        output += ";<br>";
                        last = "";
                        if (flagSingleFor) {
                            indent--;
                            flagSingleFor = false;
                        }
                        for (int i = 0; i < indent; i++) {
                            output += "    ";
                        }
                    }
                    break;
                case '{':
                    flagSingleFor = false;
                    flagFor = false;
                    flagIf = false;
                    flagWhile = false;
                    last = "";
                    //indent++;
                    output += "<b>{</b><br>";
                    for (int i = 0; i < indent; i++) {
                        output += "    ";
                    }
                    break;
                case '}':
                    last = "";
                    if (indent == 0) {
                        output += "    ";
                        output += "<b>}</b><br>";
                        indent--;
                    } else {
                        indent--;
                        if (indent < 0) {
                            indent = 0;
                        }
                        output += "<b>}</b><br>";
                        for (int i = 0; i < indent; i++) {
                            output += "    ";
                        }
                    }
                    break;
                default:
                    output += c;
                    last += c;
            }
        }
        return output;
    }

    /**
     * Helper function to write to file the content
     * @param filename
     * @param content
     * @return true if success, false otherwise
     */
    public static boolean OutputFile(String filename, String content) {
        try {
            PrintWriter pw = new PrintWriter(new FileWriter(new File(filename)));
            pw.println(content);
            pw.flush();
            pw.close();
        } catch (Exception e) {
            return false;
        }
        return true;
    }
} //--End class
```

Tableau E.15 - Le code source de la classe WebPage.java.

ANNEXE F

INTERFACE WEB DE L'OUTIL DE LA PARALLÉLISATION SEMI-AUTOMATIQUE DES PROGRAMMES SÉQUENTIELS

Nous avons implanté une interface Web permettant aux utilisateurs d'effectuer automatiquement certaines étapes de la transformation d'un programme séquentiel en un programme parallèle en utilisant la bibliothèque d'échange de messages (MPI). Cette interface, appelée *Transform to MPI* (voir la Figure F.1), permet d'effectuer les opérations suivantes : la traduction des structures de données écrites en langage *C* en structures de données MPI, la détection des boucles et l'analyse des dépendances des données associées, l'assistance pour la génération du code parallèle, le partage et la synchronisation des données (Diallo *et al.*, 2012).

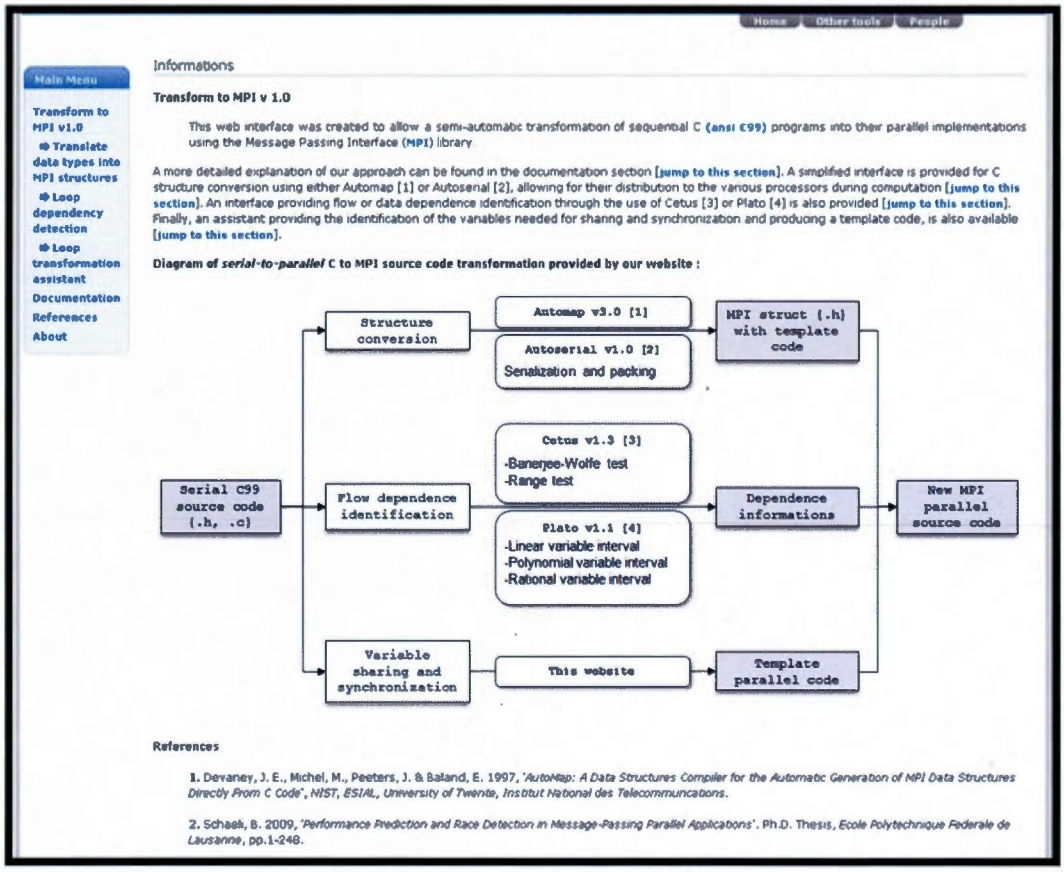


Figure F.1 - Page principale du site Web *Transform to MPI.*

F.1 Traduction des structures de données du langage C en structures de données MPI

Au niveau de la traduction des structures de données, le site Web *Transform to MPI* permet à l'utilisateur soit de sélectionner son fichier contenant les structures de données, soit de copier directement ces structures dans l'espace réservé. L'utilisateur choisit par la suite la méthode de transformation à utiliser. Une fois les structures de données identifiées et l'option de transformation choisie, le bouton *Compute* permet de lancer le processus de transformation des structures de données (voir la Figure F.2).

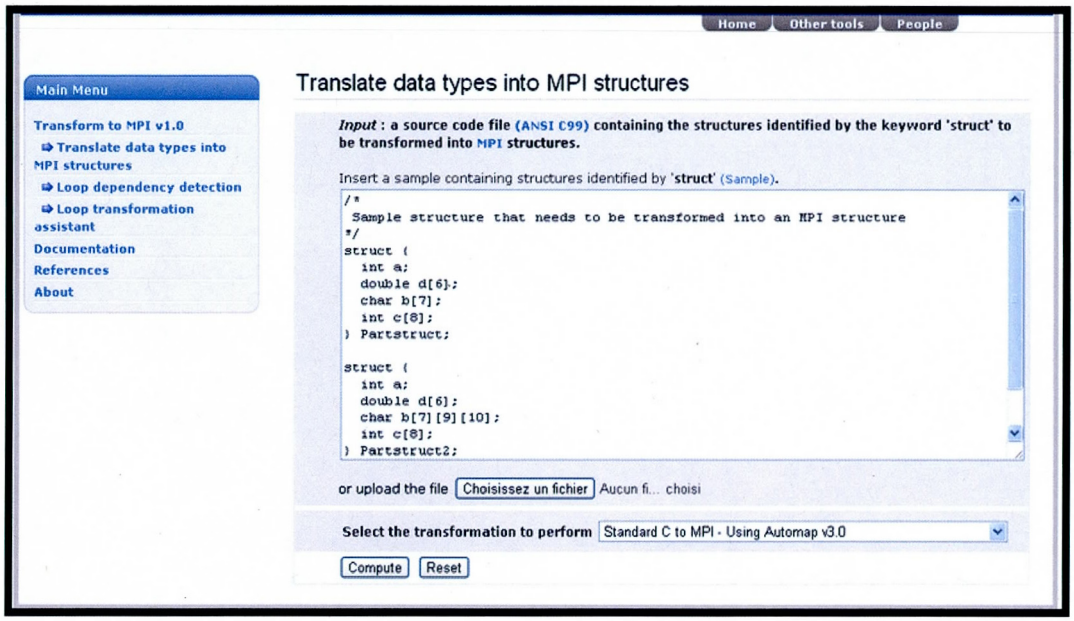


Figure F.2 - Interface de transformation des structures de données *C* en structures de données MPI.

La Figure F.3 présente les résultats du processus de transformation obtenus en utilisant l'outil *Automap* et la Figure F.4 ceux obtenus au moyen de l'outil *Autoserial*.

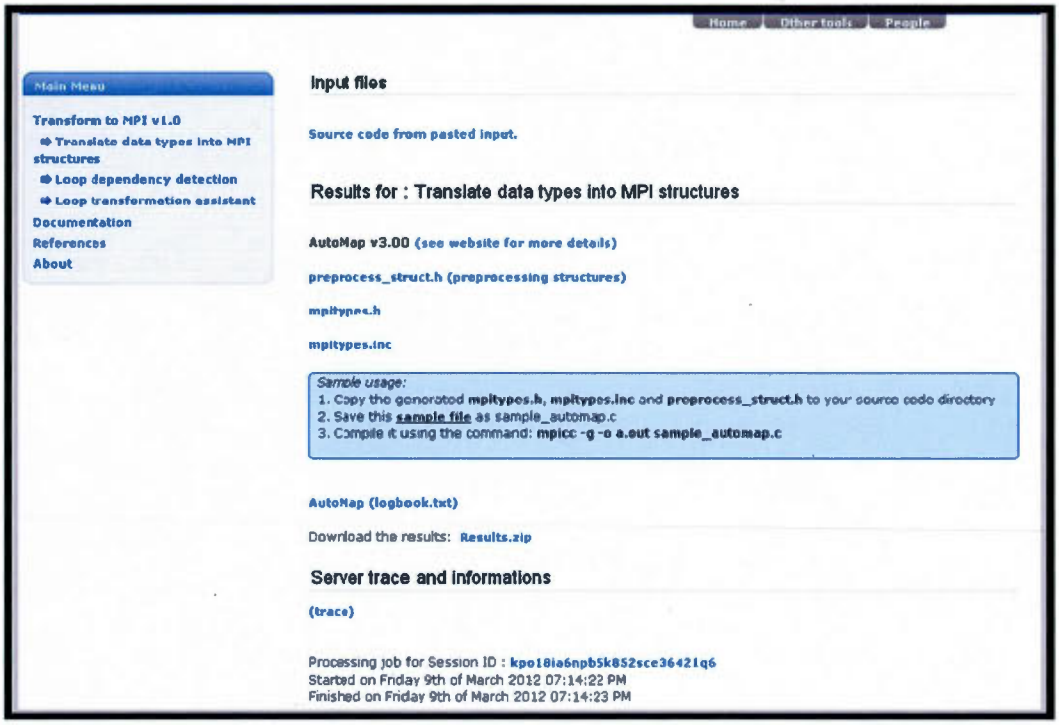


Figure F.3 - Interface présentant les résultats de la transformation des structures de données *C* en structures de données MPI en utilisant *Automap*.

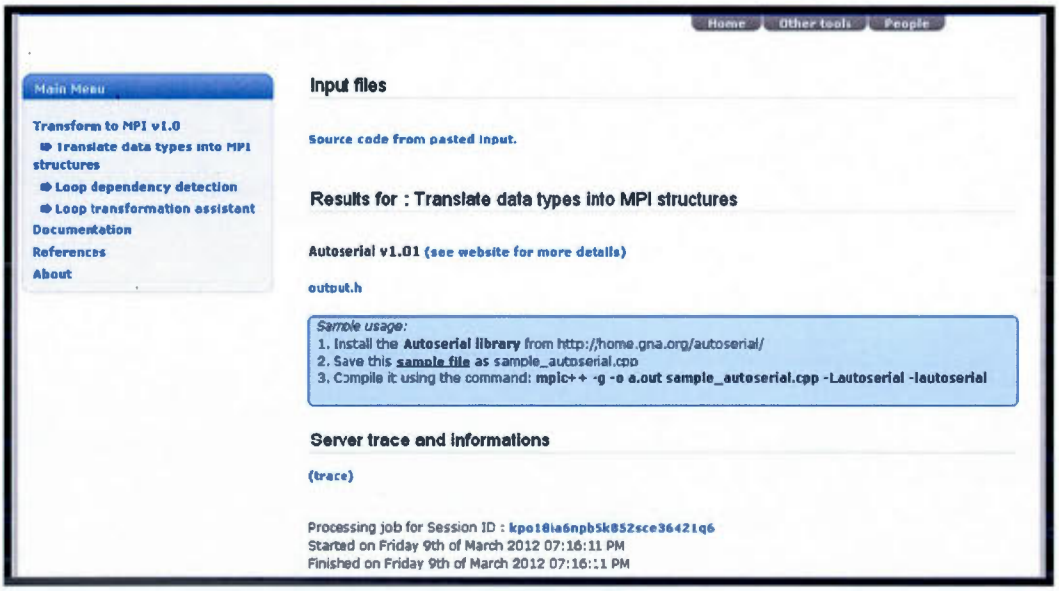


Figure F.4 - Interface présentant les résultats de la transformation des structures de données *C* en structures de données MPI en utilisant *Autoserial*.

F.2 Détection des boucles et analyse des dépendances des données

Au niveau de la détection des boucles et l'analyse des dépendances des données, la nouvelle interface (voir la Figure F.5) permet tout d'abord à l'utilisateur soit de sélectionner son fichier contenant le code source en langage *C*, soit de le copier directement dans l'espace réservé. L'mutilateur peut sélectionner ensuite la méthode d'analyse des dépendances des données à appliquer (LVI, PVI, RVI, *Banerjee-Wolfe* ou *Range test*) et puis appuyer sur l'option *Compute* pour lancer le processus (voir la Figure F.5).

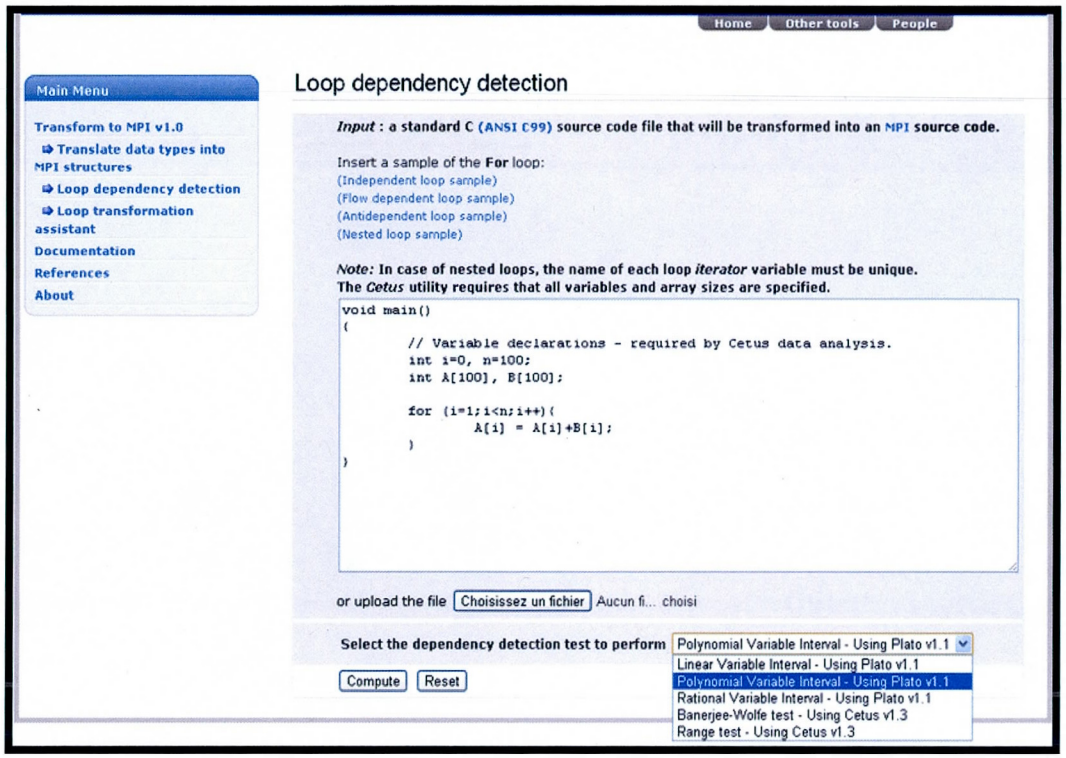


Figure F.5 - Interface pour la détection des boucles et l'analyse des dépendances des données.

La page d'affichage des résultats de l'analyse des dépendances des données associées à chaque boucle du programme est présentée sur la Figure F.6.

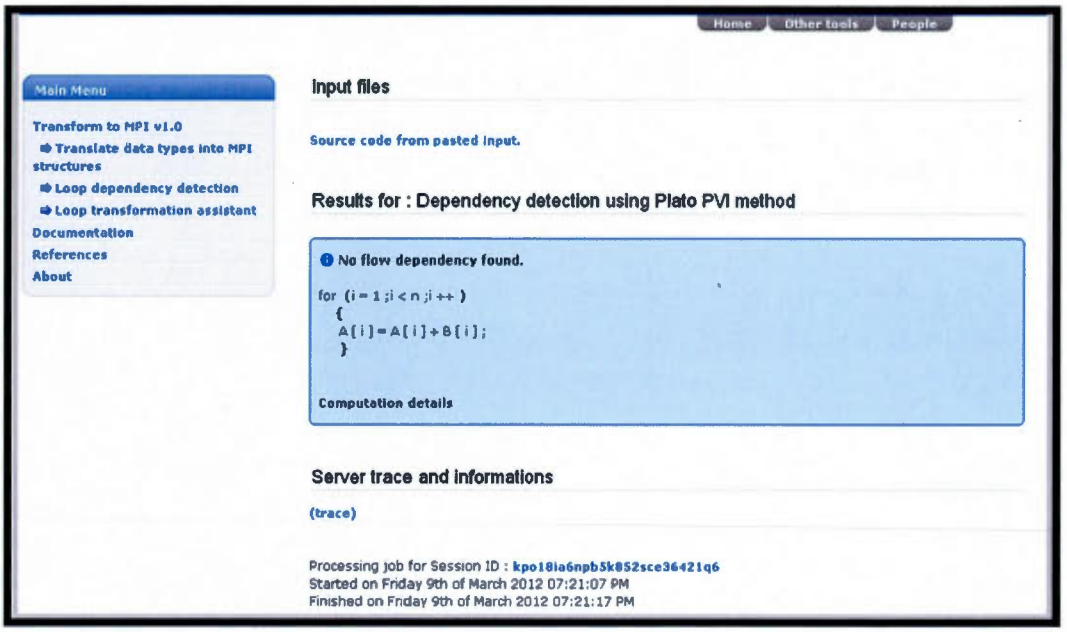


Figure F.6 - Interface présentant les résultats de la détection des dépendances des données.

F.3 Assistance la pour génération du code parallèle, le partage et la synchronisation des données

Comme lors de l'étape de la détection des dépendances des données, l'interface d'assistance permet à l'utilisateur soit de sélectionner son fichier contenant le code source en langage *C*, soit de le copier directement dans l'espace réservé. L'option *Compute* sert à lancer le processus d'assistance (voir la Figure F.7).

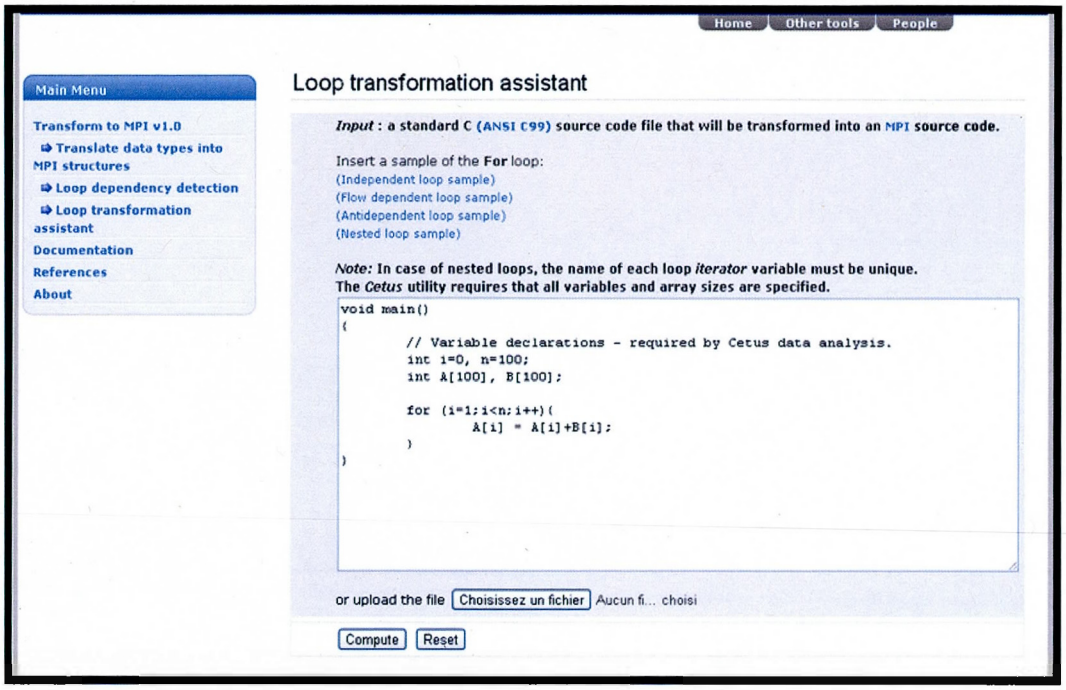


Figure F.7 - Interface de l'assistant de la génération du code parallèle.

La page d'affichage des informations fournies par l'assistant de la génération du code parallèle est présentée sur la Figure F.8.

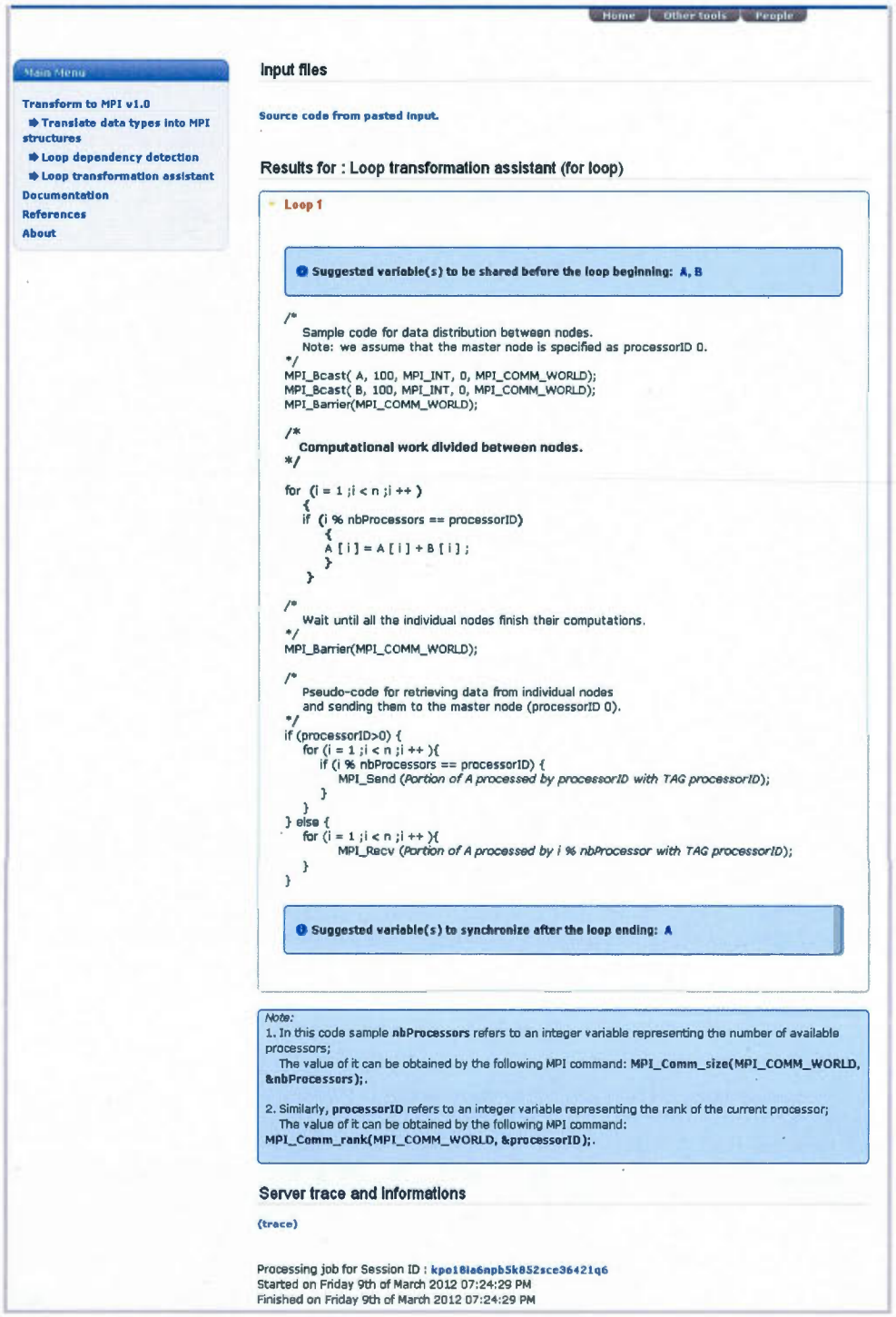


Figure F.8 - Interface présentant les résultats fournis par l'assistant de la génération du code parallèle.

GLOSSAIRE

ADN : L'acide désoxyribonucléique (souvent abrégé en ADN) est une macromolécule de poids moléculaire élevé, formée de polymères de nucléotides dont le sucre et deux désoxyriboses. L'ADN se présente sous la forme d'une double chaîne hélicoïdale dont les deux brins sont complémentaires. Il est composé d'un assemblage linéaire de nucléotides renfermant chacun dans sa structure une base azotée qui l'identifie. Il existe quatre types de nucléotides au niveau de l'ADN : l'adénine (A), la guanine (G), la cytosine (C) et la thymine (T). Ces quatre nucléotides se subdivisent en deux groupes : les purines composées de l'adénine et de la guanine; les pyrimidines composées de la cytosine et de la thymine. L'ADN est présent dans le noyau des cellules eucaryotes, dans le cytoplasme des cellules procaryotes, dans la matrice des mitochondries ainsi que dans les chloroplastes. Certains virus possèdent également de l'ADN encapsulé dans leur capside, coque constituée de protéines qui entoure et protège l'acide nucléique d'un virus. On dit que l'ADN est le support de l'hérédité. Il se transmet en partie ou en totalité lors des processus de reproduction. Il est à la base de la synthèse des protéines.

Alignement : Un alignement est une opération qui permet de comparer des séquences de nucléotides pour repérer les éléments correspondants. Cette opération est nécessaire pour toutes les comparaisons de séquences de nucléotides.

Application Programming Interface (API) : Une API a pour objet de faciliter le travail d'un programmeur en lui fournissant les outils de base nécessaires. Elle constitue une interface servant de fondement à un travail de programmation plus poussé.

Archée : Les Archées ou *Archaea* (anciennement appelés archéobactéries, du grec *archaios*, « ancien » et *backterion*, « bâton ») sont un groupe majeur de microorganismes. Elles constituent un taxon du vivant caractérisé par des cellules sans noyau et se distinguant des Eubactéries (vraies bactéries) par certains caractères biochimiques, comme la constitution de la membrane cellulaire ou le mécanisme de réplication de l'ADN.

ARN : L'acide ribonucléique (souvent abrégé ARN) est une macromolécule formée par la polymérisation de nombreux nucléotides dont le sucre et le ribose. Tout comme l'ADN, l'ARN est composé d'un assemblage linéaire de nucléotides renfermant chacun dans sa structure une base azotée qui l'identifie. Il existe quatre types de nucléotides au niveau de l'ARN : l'adénine (A), la guanine (G), la cytosine (C) et l'uracile (U). Ces quatre nucléotides se subdivisent en deux groupes : les purines composées de l'adénine et de la guanine; les pyrimidines composées de la cytosine et de l'uracile. L'ARN est présent dans le cytoplasme, les mitochondries ainsi que dans le noyau cellulaire. Il sert d'intermédiaire dans la synthèse des protéines.

Bactérie : Les bactéries ou *bacteria* appartiennent au vaste ensemble des microbes qui comprennent également les virus, les champignons et les parasites. Microorganismes invisibles à l'œil nu, les bactéries sont constituées d'une seule cellule dépourvue d'un vrai noyau. Elles contiennent un seul chromosome formé d'un long filament d'ADN.

Bootstrap : Le bootstrap est une méthode proposée à la fin des années 70 par Bradley Efron (Efron, 1979). Son but est de fournir des indications sur une statistique autre que sa valeur (dispersion, distribution, intervalles de confiance) afin de connaître la précision des estimations réalisées. Ces informations sont obtenues sans recours à de nouvelles observations. Le bootstrap peut être utilisé comme une méthode statistique pour évaluer la robustesse d'un arbre phylogénétique. Elle part de l'hypothèse que les sites évoluent de manière indépendante.

Cluster : Un cluster peut être défini comme un système informatique constitué de plusieurs unités de calcul (CPU), c'est-à-dire de micro-processeurs, cœurs ou unités centrales, qui sont reliées entre elles à l'aide d'un réseau de communication.

Compilateur : C'est un logiciel qui est chargé de traduire le code source d'un programme informatique en langage machine compréhensible par un ordinateur.

CPU : Le CPU (Central Processing Unit, ou unité centrale de traitement) ou processeur est le composant de l'ordinateur qui exécute les programmes informatiques.

Eucaryotes : Les Eucaryotes ou *eucarya* (du grec *eu*, vrai et *karuon*, noyau) comprennent 4 grands règnes du monde vivant : les animaux, les champignons, les plantes et les protistes. Ils constituent donc un très large groupe d'organismes, unis et pluricellulaires, définis par leur structure cellulaire.

FLOPS : FLOPS ou flops ou flop/s (FLoating Point Operations Per Second) est un acronyme signifiant « opérations à virgule flottante par seconde ». Les opérations à virgule flottante (additions ou multiplications) incluent toutes les opérations qui impliquent des nombres réels. Le nombre de FLOPS est une unité standard pour mesurer la vitesse d'un système informatique.

Gène : Un gène est une séquence ordonnée de nucléotides qui occupe une position précise sur un chromosome déterminé. Il constitue une information génétique dont la transmission est héréditaire, c'est-à-dire transmise par un individu à sa descendance par reproduction sexuée ou asexuée. Le gène le plus simple consiste en un segment d'acide désoxyribonucléique (ADN) codant un seul acide ribonucléique (ARN), à l'origine d'une seule protéine (en dehors de l'épissage alternatif). L'ensemble des gènes d'un individu constitue une partie de son génome.

Instruction : En informatique, on peut définir une instruction comme une commande élémentaire lue et exécutée par un processeur.

Lignée : Une lignée est le regroupement de toutes les espèces issues du même ancêtre.

Matrice de dissimilarités ou matrice de distances : Une matrice de dissimilarités est un tableau à double entrée comprenant horizontalement comme verticalement la même série d'espèces. Chacune des cases de la matrice contient une valeur de distance de dissimilarités qui sépare les deux espèces concernées. Cette valeur est égale à zéro pour deux espèces identiques.

Processus informatique : Un processus informatique est défini par un ensemble d'instructions à exécuter (un programme), un espace mémoire pour les données de travail et éventuellement, d'autres ressources, comme des descripteurs de fichiers, des ports réseau, etc.

Programme informatique : Un programme informatique est une succession d'instructions exécutables par l'ordinateur. Toutefois, l'ordinateur ne sait manipuler que des données binaires, c'est-à-dire une succession de 0 et de 1. Il est donc nécessaire d'utiliser un langage de programmation pour écrire de façon lisible, c'est-à-dire de façon compréhensible par l'humain, les instructions à exécuter par l'ordinateur.

Taxon : Groupe d'organismes vivants descendant d'un même ancêtre et qui ont tous un certain nombre de caractères communs. Plusieurs taxon forment un taxa.

RÉFÉRENCES

Addario-Berry, L., Hallett, M. & Lagergrenm, J. (2003), Towards identifying lateral gene transfer, *Proceedings of the 8th Pacific Symposium on Biocomputing 2003 (PSB03)*, pp.279-290.

Agbakpem, K. K. (2006), Transformation automatique d'arbres syntaxiques avec SableCC, *Mémoire de Maîtrise, Université du Québec à Montréal.*

Altekar, G., Dwarkadas, S., Huelsenbeck, J. P. & Ronquist, F. (2004), Parallel Metropolis-coupled Markov Chain Monte Carlo for Bayesian phylogenetic inference, *Bioinformatics*, vol.20, pp.407-415.

Altschul, S. F., Gish, W., Miller, W., Myers, E. W. & Lipman, D. J. (1990), Basic local alignment search tool, *Journal of Molecular Biology*, vol.215, no.3, pp.403-410.

Amdahl, G. M. (1967), Validity of the single processor approach to achieving large scale computing capabilities, In *Proceedings of the April 18-20 Spring Joint Computer Conference*, pp.483-485.

André, F., Le Fur, M., Mahéo, Y. & Pazat, J.-L. (1995), The Pandore data-parallel compiler and its portable runtime, In *Proceedings of the High-Performance Computing and Networking*, pp.176-183.

Arenas, M. & Posada, D. (2007), Recodon: coalescent simulation of coding DNA sequences with recombination, migration and demography, *BMC Bioinformatics*, vol.8, pp.458.

Arnold, K., Gosling, J. & Holmes, D. (2000), *The Java programming language*, Addison-Wesley Reading, MA.

Ashwin Kumar, K., Pappu, A. K., Sarath Kumar, K. & Sanyal, S. (2006), Hybrid approach for parallelization of sequential code with function level and block level parallelization, In *Proceedings of the International Symposium on Parallel Computing in Electrical Engineering*, pp.161-166.

Backus, J. (1978), The history of Fortran I, II, and III, *History of programming languages I*, pp.25-75.

Bandelt, H.-J. & Dress, A. W. M. (1989), Weak hierarchies associated with similarity measures - an additive clustering technique, *Bull. Math. Biol.*, vol.51, no.1, pp.133-166.

Bandelt, H.-J., Forster, P., Sykes, B. C. & Richards, M. B. (1995), Mitochondrial portraits of human populations using median networks, *Genetics*, vol.141, pp.743-753.

Bandelt, H.-J., Forster, P. & Rohl, A. (1999), Median-joining networks for inferring intraspecific phylogenies, *Molecular Biology and Evolution*, vol.16, pp.37-48.

Bandelt, H.-J., Macaulay, V. & Richards, M. B. (2000), Median networks: speedy construction and greedy reduction, one simulation, and two case studies from human mtDNA, *Molecular Phylogenetics and Evolution*, vol.16, pp.8-28.

Banerjee, U. (1988), An introduction to a formal theory of dependence analysis, *The Journal of Supercomputing*, vol.2, no.2, pp.133-149.

Banerjee, P., Chandy, J. A., Gupta, M., Hodges, E. W., Holm, J. G., Lain, A., Palermo, D. J., Ramaswamy, S. & Su, E. (1995), The PARADIGM compiler for distributed-memory multicomputers, *Computer*, vol.28, no.10, pp.37-47.

Barnes, J. (2006), *Programming in ADA 2005*, Addison-Wesley.

Barthélémy, J. P. & Guénoche, A. (1991), *Trees and proximity representations*, John Wiley Sons.

Beiko, R. & Hamilton, N. (2006), Phylogenetic identification of lateral genetic transfer events, *BMC Evolutionary Biology*, vol.6, no.1, pp.15.

Benkner, S. (1999a), VFC: the Vienna Fortran compiler, *Scientific Programming*, vol.7, no.1, pp.67-81.

Benkner, S. (1999b), HPF+: High Performance Fortran for advanced scientific and engineering applications, *Future Generation Computer Systems*, vol.15, no.3, pp.381-391.

Benkner, S. & Zima, H. (1999), Compiling high performance Fortran for distributed-memory architectures, *Parallel Computing*, vol.25, no.13-14, pp.1785-1825.

Beyer, W. A., Stein Temple, F. & Myron, L. (1974), A molecular sequence metric and evolutionary trees, *Mathematical Biosciences*, vol.19, no.1-2, pp.9-25.

Blume, W. & Eigenmann, R. (1994), The range test: a dependence test for symbolic, non-linear expressions. *In Supercomputing*, pp.528-37.

Boc, A. & Makarenkov, V. (2003), New efficient algorithm for detection of horizontal gene transfer events, *Algorithms in Bioinformatics*, pp.190-201.

Boc, A., Philippe, H. & Makarenkov, V. (2010), Inferring and validating horizontal gene transfer events using bipartition dissimilarity, *Systematic Biology*, vol.59, no.2, pp.195.

Boc, A., Diallo, Al. B. & Makarenkov, V. (2011), Un nouvel algorithme pour la détection de transferts horizontaux de gènes partiels entre les espèces et pour la classification des transferts inférés, *Actes des 18-èmes Rencontres de la Société Francophone de Classification*, Orléans, France, pp.25-28.

Boc, A., Diallo, Al. B. & Makarenkov, V. (2012), T-Rex : a web server for inferring, validating and visualizing phylogenetic trees and networks. Accepté pour publication dans *Nucleic Acids Research.*

Bondhugula, U., Hartono, A., Ramanujam, J. & Sadayappan, P. (2008), Pluto: A practical and fully automatic polyhedral program optimization system, in *Proceedings of the*

ACM SIGPLAN 2008 Conference on Programming Language Design and Implementation (PLDI 08).

Bordewich, M. & Semple, C. (2004), On the Computational Complexity of the Rooted Subtree Prune and Regraft Distance, *Annals of Combinatorics*, vol.8, pp.409-423.

Brandes, T. (1993), 1 Adaptor: A Compilation System for Data Parallel Fortran Programs, *Christoph W. Kessler (Ed.)*, pp.85.

Bryant, D. & Waddell, P. (1998), Rapid evaluation of least-squares and minimum-evolution criteria on phylogenetic trees, *Molecular Biology and Evolution*, vol.15, no.10, pp.1346-1359.

Bryant, D., Tsang, J., Kearney, P. & Li, M. (2000), Computing the quartet distance between evolutionary trees, In *Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*, pp.285-286.

Bryant, D. & Moulton, V. (2004), Neighbor-net: an agglomerative method for the construction of phylogenetic networks, *Molecular Biology and Evolution*, vol.21, no.2, pp.255-265.

Butterfield, A., Vedagiri, V., Lang, E., Lawrence, C., Wakefield, M. J., Isaev, A. & Huttley, G. A. (2004), PyEvolve: a toolkit for statistical modelling of molecular evolution, *BMC Bioinformatics*, vol.5, no.1, pp.1-12.

Cavalli-Sforza, L. L. & Edwards, A. W. (1967), Phylogenetic analysis. Models and estimation procedures, *American Journal of Human Genetics*, vol.19, no.3 Pt 1, pp.233.

Chandra, R., Dagum, L., Kohr, D., Maydan, D., McDonald, J. & Menon, R. (2001), *Parallel programming in OpenMP*, Morgan Kaufmann.

Charleston, M. A. (1998), Jungles: a new solution to the host/parasite phylogeny reconciliation problem, *Mathematical Biosciences*, vol.149, no.2, pp.191-223.

Cheetham, J., Dehne, F., Pitre, S., Rau-Chaplin, A. & Taillon, J. P. (2003), Parallel ClustalW for PC clusters, In *Proceedings of the 2003 International Conference on Computational Science and Its Applications*, pp.300–309

Chen, S. H., Su, S.Y., Lo, C. Z., Chen, K. H., Huang, T. J., Kuo, B. H. & Lin, C. Y. (2009), PALM: a paralleled and integrated framework for phylogenetic inference with automatic likelihood model selectors, *PLoS One*, vol.4, no.12, pp.e8116.

Chevance, R. J. (2000), *Serveurs multiprocesseurs, clusters et architecture paralleles*, Eyrolles.

Collignon, B., Schulz, R., Smith, J. C. & Baudry, J. (2011), Task-parallel message passing interface implementation of Autodock4 for docking of very large databases of compounds using high-performance super-computers, *Journal of Computational Chemistry*, vol.32, no.6, pp.1202-1209.

Dagum, L. & Menon, R. (1998), OpenMP: an industry standard API for shared-memory programming, *IEEE Computational Science Engineering*, vol.5, no.1, pp.46-55.

Darema, F., George, D. A., Norton, V. A. & Pfister, G. F. (1988), A single-program-multiple-data computational model for EPEX/FORTRAN, *Parallel Computing*, vol.7, no.1, pp.11-24.

Darling, A. E., Carey, L., Feng & W. (2003), The Design, Implementation, and Evaluation of mpiBLAST, In *Proceedings of ClusterWorld.*

Darriba, D., Taboada, G. L., Doallo, R. & Posada, D. (2011), ProtTest 3: fast selection of best-fit models of protein evolution, *Bioinformatics*, vol.27, no.8, pp.1164.

Darwin, C. (1958), *The origin of species*, Hayes Barton Press.

Dave, C., Bae, H., Min, S.-J., Lee, S., Eigenmann, R., & Midkiff, S. (2009), Cetus: A Source-to-Source Compiler Infrastructure for Multicores, *Computer*, vol.42, no.12, pp.36-42.

de Araujo Macedo, E., Magalhaes Alves de Melo, A. C., Pfitscher, G. H. & Boukerche, A. (2011), Hybrid MPI/OpenMP Strategy for Biological Multiple Sequence Alignment with DIALIGN-TX in Heterogeneous Multicore Clusters, In *Proceedings of the International Parallel and Distributed Systems Workshops* (IPDPSW), pp.418-425.

Delwiche, C. F. & Palmer, J. D. (1996), Rampant Horizontal Transfer and Duplication of Rubisco Genes in Eubacteria and Plastids, *Mol. Biol. Evol.*, vol.13, pp.873-882.

Denamur, E., Lecointre, G., Darlu, P., Tenaillon, O., Acquaviva, C., Sayada, C., Sunjevaric, I., Rothstein, R., Elion, J., Taddei, F. & Radman, M. (2000), Evolutionary implications of the frequent horizontal transfer of mismatch repair genes, *Cell*, vol.103, no.5, pp.711-721.

Desper, R. & Gascuel, O. (2002), Fast and accurate phylogeny reconstruction algorithms based on the minimum-evolution principle, *Journal of computational biology*, vol.9, no.5, pp.687-705.

Devaney, J. E., Michel, M., Peeters, J. & Baland, E. (1997), AutoMap: A Data Structures Compiler for the Automatic Generation of MPI Data Structures Directly From C Code, *NIST, ESIAL, University of Twente, Institut National des Telecommuncations.*

Diallo, Al. B., Lord, E. & Makarenkov, V. (2012), A new web server for parallelization of bioinformatics algorithms. Soumis.

Diday, E. & Bertrand, P. (1986), An extension of hierarchical clustering: the pyramidal representation. In *Pattern Recognition in Practice* (Gelsema, E.S. and Kanal, L.N. eds.), North-Holland, pp.411-424.

Doolittle, W. F. (1999), Phylogenetic classification and the universal tree, *Science*, vol.284, no.5423, pp.2124.

Doolittle, W. F., Boucher, Y., Nesbo, C. L., Douady, C. J., Andersson, J. O. & Roger, A. J. (2003), How big is the iceberg of which organellar genes in nuclear genomes are but the tip?, *Philosophical Transactions of the Royal Society of London. Series B: Biological Science*, vol.358, no.1429, pp.39-58.

Drummond, A. J. & Rambaut, A. (2007), BEAST: Bayesian evolutionary analysis by sampling trees, *BMC Evolutionary Biology*, vol.7, no.1, pp.214.

Duncan, R. (1990), A survey of parallel computer architectures, *Computer*, vol.23, no.2, pp.5-16.

Edwards, A. & Cavalli-Sforza, L. L. (1963), The reconstruction of evolution, *Heredity*, vol.18.

Edwards, A. & Cavalli-Sforza, L. L. (1964), Reconstruction of phylogenetic trees, In *Phenetic and Phylogenetic Classification*, Systematics Association, London, pp.67-76.

Efron, B. (1979), Bootstrap methods: another look at the jackknife, *The Annals of Statistics*, vol.7, no.1, pp.1-26.

Excoffier, L. & Smouse, P. E. (1994), Using allele frequencies and geographic subdivision to reconstruct gene trees within a species: molecular variance parsimony, *Genetics*, vol.136, no.1, pp.343-359.

Farris, J. S. (1970), Methods for computing Wagner trees, *Systematic Biology*, vol.19, no.1, pp.83.

Felsenstein, J. (1981), Evolutionary trees from DNA sequences: a maximum likelihood approach, *Journal of Molecular Evolution*, vol.17, no.6, pp.368-376.

Felsenstein, J. (1989), PHYLIP - Phylogeny Inference Package (Version 3.2), *Cladistics*, vol.5, pp.164-166.

Felsenstein, J. (2004), Inferring phylogenies, *Sunderland, Massachusetts: Sinauer Associates*.

Fernando, R. (2004), *GPU Gems: Programming Techniques, Tips and Tricks for Real-Time Graphics*, Pearson Higher Education.

Fitch, W. M. & Margoliash, E. (1967), Construction of phylogenetic trees, *Science*, vol.155, no.760, pp.279-284.

Fitch, W. M. (1971), Toward defining the course of evolution: minimum change for a specific tree topology, *Systematic Biology*, vol.20, no.4, pp.406.

Fitch, W. M. (1997), Networks and viral evolution, *Journal of Molecular Evolution*, vol.44, pp.65–75.

Flynn, M. J. (1972), Some computer organizations and their effectiveness, *IEEE Transactions on Computers,* vol.100, no.9, pp.948-960.

Foulds, L. R., Hendy, M. D. & Penny, D. (1979), A graph theoretic approach to the development of minimal phylogenetic trees, *Journal of Molecular Evolution*, vol.13, no.2, pp.127-149.

Fox, G., Hiranandani, S., Kennedy, K., Koelbel, C., Kremer, U., Tseng, C. W. & Wu, M. Y. (1990), Fortran D language specification. *Tech. Rep. TR 90-141, Dept. of Computer Science Rice University*.

Gagnon, E. M. & Hendren, L. J. 1998, SableCC, An Object-Oriented Compiler Framework, In *Proceedings of the Technology of Object-Oriented Languages and Systems*, pp.140-154.

Gambette, P. & Huson, D. H. (2008), Improved layout of phylogenetic networks, *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol.5, no.3, pp.472-479.

Gasper, P., Herbst, C., McCough, J., Rickett, C. & Stubbendieck, G. (2003), Automatic parallelization of sequential C code, *Midwest Instruction and Computing Symposium, Duluth*, MN, USA.

Gauss, C.F. (1811), Disquisitio de Elementis Ellipticis Palladis, English translation of extract in pp.148-155 of Trotter, H . F. (1957), Gauss's Work (1803-26) on the Theory of Least Squares, *Technical Report 5, Statistical Techniques Research Group,* Princeton University, A translation of Méthodes des Moindres Carrés, the authorised French translation of Gauss's writings on least squares by J. Bertrand (1855), Paris: Mallet-Bachelier.

Geist, A., Beguelin, A., Dongarra, J., Jiang, W., Manchek, R. & Sunderam, V. S. (1994), PVM: Parallel Virtual Machine: a user's guide and tutorial for network parallel computing, *MIT Press* (Cambridge, Mass.).

Germain-Renaud, C. & Sansonnet, J. (1991), *Les ordinateurs massivement parallèles*, Armand Colin.

Ghemawat, S. & Dean, J. (2004), MapReduce: Simplified data processing on large clusters, In *Proceedings of the 6th Symposium on Operating System Design and Implementation, San* Francisco, CA, USA.

Gogarten, J. P., Doolittle, W. F. & Lawrence, J. G. (2002), Prokaryotic evolution in light of gene transfer, *Molecular Biology and Evolution*, vol.19, no.12, pp.2226-2238.

Grama, A., Gupta, A., Karypis, G. & Kumar, V. (2003), *Introduction to parallel computing*, Addison Wesley Longman.

Gropp, W., Lusk, E., Doss, N. & Skjellum, A. (1996), A high-performance, portable implementation of the MPI message passing interface standard, *Parallel computing*, vol.22, no.6, pp.789-828.

Gropp, W., Thakur, R. & Lusk, E. (1999), Using MPI-2: Advanced features of the message passing interface, *MIT press*.

Grundmann, T., Ritt, M., & Rosenstiel, W. (2000), TPO++: An Object-Oriented Message-Passing Library in C++, In *Proceedings of the Proceedings of the 2000 International Conference on Parallel Processing* (ICPP '00), IEEE Computer Society, pp.43-50.

Guindon, S. & Gascuel, O. (2002), Efficient biased estimation of evolutionary distances when substitution rates vary across sites, *Molecular Biology and Evolution*, vol.19, no.4, pp.534-543.

Guindon, S. & Gascuel, O. (2003), A simple, fast, and accurate algorithm to estimate large phylogenies by maximum likelihood, *Systematic biology*, vol.52, no.5, pp.696-704.

Guindon, S., Dufayard, J. F., Lefort, V., Anisimova, M., Hordijk, W. & Gascuel, O. (2010), New algorithms and methods to estimate maximum-likelihood phylogenies: assessing the performance of PhyML 3.0, *Systematic biology*, vol.59, no.3, pp.307-321.

Haeckel, E. H. (1866), *Generelle Morphologie der Organismen: allgemeine Grundzüge der organischen Formen-Wissenschaft, mechanisch begründet durch die von Charles Darwin reformirte Descendenz-Theorie*, vol.2, G. Reimer.

Haeckel, E. H. (1874), *Histoire de la création des êtres organisés d'après les lois naturelles*, C. Reinwald et cie.

Hall, M. W., Anderson, J. M., Amarasinghe, S. P., Murphy, B. R., Liao, S. W. & Bu, E. (1996), Maximizing multiprocessor performance with the SUIF compiler, *Computer*, vol.29, no.12, pp. 84-89.

Hallett, M. T. & Lagergren, J. (2001), Efficient algorithms for lateral gene transfer problems, In *Proceedings of the fifth annual international conference on Computational biology*, ACM, pp.149-156.

Hallett, M., Lagergren, J. & Tofigh, A. (2004), Simultaneous identification of duplications and lateral transfers, In *Proceedings of the eighth annual international conference on research in computational biology* (P.E. Bourne et D. Gusfield Eds.), ACM, San Diego, pp.347-356.

Hamming, R. W. (1950), Error detecting and error correcting codes, *Bell System Technical Journal*, vol.29, no.2, pp.147-160.

Hein, J. (1990), Reconstructing evolution of sequences subject to recombination using parsimony, *Mathematical Biosciences*, vol.98, no.2, pp.185-200.

Hein, J. (1993), A heuristic method to reconstruct the history of sequences subject to recombination, *Journal of Molecular Evolution*, vol.36, no.4, pp.396-405.

Hein, J., Jiang, T., Wang, L. & Zhang, K. (1996), On the complexity of comparing evolutionary trees, *Discrete Applied Mathematics*, vol.71, no.1-3, pp.153-169.

Hendren, L. J. & Nicolau, A. (1990), Parallelizing programs with recursive data structures, *IEEE Transactions on Parallel and Distributed Systems*, vol.1, no.1, pp.35-47.

Hillson, R. & Iglewski, M. (2000), C++2MPI: a software tool for automatically generating MPI datatypes from C++ classes, In *Proceedings of the Parallel Computing in Electrical Engineering*, pp.13-17.

Hockney, R. (1988), Classification and evaluation of parallel computer systems, In *Proceedings of Parallel Computing in Science and Engineering*, pp.13-25.

Hollingshead, S. K., Becker, R. & Briles, D. E. (2000), Diversity of PspA: mosaic genes and evidence for past recombination in Streptococcus pneumoniae, *Infection and immunity*, vol.68, no.10, pp.5889-5900.

Huelsenbeck, J. P. & Ronquist, F. (2001), MRBAYES: Bayesian inference of phylogenetic trees, *Bioinformatics*, vol.17, no.8, pp.54-755.

Hughes, S. & Zmievski, A. (2001), *PHP developer's cookbook*, Sams Publishing.

Hurteau, G., Van Dongen, V. & Gao, G. (1994), Overview of EPPP-an Environment for Portable Parallel Programming, In *Proceedings of the Supercomputing Symposium*, vol.94, pp.119-127.

Huson, D. H. & Bryant, D. (2006), Application of phylogenetic networks in evolutionary studies, *Molecular Biology and Evolution*, vol.23, no.2, pp.254-267.

Jain, R., Rivera, M. C. & Lake, J. A. (1999), Horizontal gene transfer among genomes: the complexity hypothesis, In *Proceedings of the National Academy of Sciences*, vol.96, no.7, pp.3801- 3806.

Johnson, T. A., Lee, S., Fei, L., Basumallik, A., Upadhyaya, G., Eigenmann, R. & Midkiff, S. (2005), Experiences in Using Cetus for Source-to-Source Transformations, *Lecture Notes in Computer Science*, vol.3602, pp.1-14.

Jones, D. T., Taylor, W. R. & Thornton, J. M. (1992), The rapid generation of mutation data matrices from protein sequences, *Computer Applications in the Biosciences: CABIOS*, vol.8, no.3, pp.275.

Katoh, K. & Toh, H. (2010), Parallelization of the MAFFT multiple sequence alignment program, *Bioinformatics*, vol.26, no.15, pp.1899-1900.

Keays, R. & Rakotonirainy, A. (2003), Context-oriented programming, In *Proceedings of the 3rd ACM international workshop on Data engineering for wireless and mobile access*, ACM, pp.9-16.

Kernighan, B. W. & Ritchie, D. M. (1988), *The C Programming Language*, Prentice Hall.

Kidd, K. K. & Sgaramella-Zonta, L. A. (1971), Phylogenetic analysis: concepts and methods, *American Journal of Human Genetics*, vol.23, no.3, pp.235-252.

Kim, J. & Warnow, T. (1999), Tutorial on phylogenetic tree estimation, *Intelligent Systems for Molecular Biology,* Heidelberg.

Koonin, E. V. (2003), Horizontal gene transfer: the path to maturity, *Molecular microbiology*, vol.50, no.3, pp.725-727.

Kuhner, M. K. & Felsenstein, J. (1994), A simulation comparison of phylogeny algorithms under equal and unequal evolutionary rates, *Molecular Biology and Evolution*, vol.11, no.3, pp.459-468.

Kyriakopoulos, K. (2007), Advancing data dependence analysis in practice, *Thesis, University of Texas at San Antonio*.

Lapointe, F.-J. (2000), How to account for reticulation events in phylogenetic analysis: A comparison of distance-based methods, *Journal of Classification*, vol.17,no.2, pp.175-184.

Lawrence, J. G. & Ochman H. (1997), Amelioration of bacterial genomes: rates of change and exchange, *Journal of Molecular Evolution*, vol.44, no.4, pp.383-397.

Lawrence, J. G. (1999), Gene transfer, speciation, and the evolution of bacterial genomes, *Current Opinion in Microbiology*, vol.2, no.5, pp.519-523.

Lee, S., Min, S. J. & Eigenmann, R. (2009), OpenMP to GPGPU: a compiler framework for automatic translation and optimization, *ACM SIGPLAN Notices*, vol.44, no.4, pp.101-110.

Legendre, P. (2000), Special section on reticulate evolution, *Journal of Classification*, vol.17, pp.153-195.

Legendre, P. & Makarenkov, V. (2002), Reconstruction of biogeographic and evolutionary networks using reticulograms, *Systematic Biology*, vol.51, no.2, pp.199-216.

Li, K. B. (2003), ClustalW-MPI: ClustalW analysis using distributed and parallel computing, *Bioinformatics*, vol.19, no.12, pp.1585-1586.

Li, S., Pearl, D. K. & Doss, H. (2000), Phylogenetic tree construction using Markov chain Monte Carlo, *Journal of the American Statistical Association*, vol.95, no.450, pp. 493-508.

Li, W. H. & Ere-Walker, A. (1997), *Molecular evolution*, Sinauer Associates Sunderland.

Lin, H., Ma, X., Chandramohan, P., Geist, A. & Samatova, N. (2005), Efficient data access for parallel BLAST, In *Proceedings of the 19th IEEE International Parallel & Distributed Systems,* Denver.

Lin, H., Ma, X., Feng, W. & Samatova, N. (2010), Coordinating Computation and I/O in Massively Parallel Sequence Search, In *Proceedings of the IEEE International Parallel & Distributed Systems*, vol.22, pp.529-543.

Liu W., B. Schmidt, G., Voss, G. & Müller-Wittig, W. (2007), Streaming algorithms for biological sequence alignment on GPUs, In *Proceedings of the IEEE Transactions on Parallel and Distributed Systems*, vol.18, no.9, pp.1270-1281.

Liu, Y., Maskell, D. L. & Schmidt, B. (2009), CUDASW++: optimizing Smith-Waterman sequence database searches for CUDA-enabled graphics processing units, *BMC Research Notes*, vol.2, no.1, pp.73.

Liu, Y., Schmidt, B., Liu, W., Maskell, D. L. (2010a), CUDA-MEME: accelerating motif discovery in biological sequences using CUDA-enabled graphics processing units, *Pattern Recognition Letters*, vol.31, no.14, pp.2170-2177.

Liu, Y., Schmidt, B. & Maskell, D. L. (2010b), CUDASW++2.0: enhanced Smith-Waterman protein database search on CUDA-enabled GPUs based on SIMT and virtualized SIMD abstractions, *BMC Research Notes*, vol.3, no.1, pp.93.

Liu, Y., Schmidt, B. & Maskell, D. L. (2010c), MSAProbs: multiple sequence alignment based on pair hidden Markov models and partition function posterior probabilities, *Bioinformatics*, vol.26, no.16, pp.1958-1964.

Liu, Y., Schmidt, B. & Maskell, D. L. (2011a), An ultrafast scalable many-core motif discovery algorithm for multiple GPUs, In *Proceedings of the 10th IEEE International Workshop on High Performance Computational Biology* (HiCOMB 2011), pp.428-434.

Liu, W., Schmidt, B. & Müller-Wittig, W. (2011b), CUDA-BLASTP: accelerating BLASTP on CUDA-enabled graphics hardware, *IEEE/ACM Transactions on Computational Biology and Bioinformatics* (TCBB), vol.8, no.6, pp.1678-1684.

MacLeod, D., Charlebois, R., Doolittle, F. & Bapteste, E. (2005), Deduction of probable events of lateral gene transfer through comparison of phylogenetic trees by recursive consolidation and rearrangement, *BMC evolutionary biology*, vol.5, no.1, pp.27.

Maddison, W. P. (1997), Gene trees in species trees, *Systematic Biology*, vol.46, no.3, pp.523-5326.

Makarenkov, V. & Leclerc, B. (1999), An algorithm for the fitting of a tree metric according to a weighted least-squares criterion, *Journal of Classification*, vol.16, no.1, pp.3-26.

Makarenkov, V. (2001), T-REX: reconstructing and visualizing phylogenetic trees and reticulation networks, *Bioinformatics*, vol.17, no.7, pp.664-668.

Makarenkov, V. & Legendre, P. (2004), From a phylogenetic tree to a reticulated network, *Journal of Computational Biology*, vol.11, no.1, pp.195-212.

Makarenkov, V., Boc, A., Delwiche, C. F., Diallo, Ab. B. & Philippe, H. (2006), New efficient algorithm for modeling partial and complete gene transfer scenarios, *Data Science and Classification*, pp. 341-349.

Makarenkov, V., Boc, A. & Diallo, Al. B. (2007), La dissimilarité de bipartitions et son utilisation pour détecter les transferts horizontaux de gènes, *Actes des 14-emes Rencontres de la Société Francophone de Classification*, ENST de Paris, France, pp.90-93.

Makarenkov, V., Boc, A., Diallo, Al. B. & Diallo, Ab. B. (2008), Algorithms for detecting complete and partial horizontal gene transfers: Theory and practice, in Data Mining and Mathematical Programming, In *CRM Proceedings and AMS Lecture Notes*, pp.159-179.

Manavski, S. A. & Valle, G. (2008), CUDA compatible GPU cards as efficient hardware accelerators for Smith-Waterman sequence alignment, *BMC Bioinformatics*, vol.9, no.2, pp.s10.

Mardis, E. R. (2008), The impact of next-generation sequencing technology on genetics, *Trends in Genetics*, vol.24, no.3, pp.133-141.

Margush, T. & McMorris, F. R. (1981), Consensusn-trees, *Bulletin of Mathematical Biology*, vol.43, no.2, pp.239-244.

Massingill, B. L., Mattson, T. G. & Sanders, B. A. (1999), Patterns for parallel application programs, In *Proceedings of the Sixth Pattern Languages of Programs Workshop* (PLoP 1999).

Massingill, B. L., Mattson, T. G. & Sanders, B. A. (2000), Patterns for finding concurrency for parallel application programs, In *Proceedings of the Seventh Pattern Languages of Programs Workshop* (PLoP 2000).

Matte-Tailliez, O., Brochier, C., Forterre, P. & Philippe, H. (2002), Archaeal phylogeny based on ribosomal proteins, *Molecular biology and evolution*, vol.19, no.5, pp.631-639.

Mattson, T. G., Sanders, B. A. & Massingill, B. (2005), *Patterns for parallel programming*, Addison-Wesley Professional.

McCandless, B. C., Squyres, J. M. & Lumsdaine, A. (1996), Object Oriented MPI (OOMPI): a class library for the Message Passing Interface, In *Proceedings of the second MPI Developer's Conference*, pp.87-94.

McGraw, J. R. & Axelrod, T. S. (1987), Exploiting multiprocessors: issues and options, *Programming parallel processors* (Addison-Wesley Longman Publishing), pp.7-25.

Mirkin, B. G., Fenner, T. I., Galperin, M. Y. & Koonin, E. V. (2003), Algorithms for computing parsimonious evolutionary scenarios for genome evolution, the last universal common ancestor and dominance of horizontal gene transfer in the evolution of prokaryotes, *BMC Evolutionary Biology.*, vol.3, no.2, pp.2.

Mirkin, B., Muchnik, I. & Smith, T. F. (1995), A biologically consistent model for comparing molecular phylogenies, *Journal of computational biology*, vol.2, no.4, pp.493-507.

Nakhleh, L., Ruths, D. & Wang, L. S. (2005), RIATA-HGT: A fast and accurate heuristic for reconstructing horizontal gene transfer, *Computing and Combinatorics*, pp.84-93.

Nelson, G. (1979), Cladistic analysis and synthesis: Principles and definitions, with a historical note on Adanson's Familles des Plantes (1763-1764), *Systematic Zoology*, vol.28, no.1, pp.1-21.

Nelson, K. E., Clayton, R. A., Gill, S. R., Gwinn, M. L., Dodson, R. J., Haft, D. H., Hickey, E. K., Peterson, J. D., Nelson, W. C., Ketchum, K. A. & others. (1999), Evidence for lateral gene transfer between Archaea and bacteria from genome sequence of Thermotoga maritima, *Nature*, vol.399, no.6734, pp.323-329.

Neyman, J. 1(971), Molecular studies of evolution: A source of novel statistical problems, *Statistical decision theory and related topics*, pp.1-27.

Nvidia, C. (2010), *Programming guide version 3.1.1*, Nvidia Corporation.

Oracle White Paper (2010), *Parallel Programming with Oracle® Developer Tools.*

Pacheco, P. (1996), *Parallel Programming with MPI*, Morgan Kaufmann.

Padua, D. A., Eigenmann, R., Hoeflinger, J., Petersen, P., Tu, P., Weatherford, S. & Faigin, K. (1993), Polaris: A new-generation parallelizing compiler for MPPs, In *CSRD Rept. No. 1306. Univ. of Illinois at Urbana-Champaign.*

Page, R. D. (1994), Maps between trees and cladistic analysis of historical associations among genes, organism and areas, *Systematic Biology*, vol.43, no.1, pp.58-77.

Page, R. D. & Charleston, M. A. (1997), From gene to organismal phylogeny: reconciled trees and gene tree/species tree problem, *Molecular Phylogenetics and Evolution*, vol.7, no.2, pp.231-240.

Page, R. D. & Charleston, M. A. (1998), Trees within trees: phylogeny and historical associations, *Trends in Ecology and Evolution*, vol.13, no.9, pp.356-359.

Paraskevis, D., Deforche, K., Lemey, P., Magiorkinis, G., Hatzakis, A. & Vandamme, A. M. (2005), SlidingBayes: exploring recombination using a sliding window approach based on Bayesian phylogenetic inference, *Bioinformatics*, vol.21, no.7, pp.1274-1275.

Pfeiffer, W. & Stamatakis, A. (2010), Hybrid MPI/Pthreads parallelization of the RAxML phylogenetics code, In *Proceedings of the IEEE International Parallel & Distributed Systems*, pp.1-8.

Pond, K. S. L., Frost, S. D. & Muse, S. V. (2005), HyPhy: hypothesis testing using phylogenies, *Bioinformatics*, vol.21, no.5, pp.676.

Pond, K. S. L., Posada, D., Gravenor, M. B., Woelk C. H. & Frost, S. D. (2006), GARD: a genetic algorithm for recombination detection, *Bioinformatics*, vol.22, no.24, pp.3096-3098.

Powell, T. A., Young, B. & Hansen, B. 1999, *HTML: the complete reference*, Osborne/McGraw-Hill.

Psarris, K. & Kyriakopoulos, K. (1999), Data dependence testing in practice, In *Proceedings of the Parallel Architectures and Compilation Techniques,* pp. 264-273.

Psarris, K. & Kyriakopoulos, K. (2004), An experimental evaluation of data dependence analysis techniques, In *Proceedings of the IEEE Parallel and Distributed Systems*,vol.15, no.3, pp. 196-213.

Quinn, M. J. (2004), *Parallel programming in C with MPI and OpenMP*, McGraw-Hill Science/Engineering/Math.

Rambaut, A. & Grassly, N. C. (1997), Seq-Gen: an application for the Monte Carlo simulation of DNA sequence evolution along phylogenetic trees, *Computer applications in the biosciences: CABIOS*, vol.13, no.3, pp.235-238.

Rannala, B. & Yang, Z. (1996), Probability distribution of molecular evolutionary trees: a new method of phylogenetic inference, *Journal of Molecular Evolution*, vol.43, no.3, pp.304-311.

Rannala, B. (1997), Gene genealogy in a population of variable size, *Heredity*, vol.78, no.4, pp.417-423.

Ray, S. C. (1998), Simplot, *Division of Infectious Diseases, Johns Hopkins University School of Medicine, Baltimore,* USA.

Renault, E. (2007), Extended MPICC to generate MPI derived datatypes from C datatypes automatically, In *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, pp.307-314.

Rieseberg, L. H. & Morefield, J. D. (1995), Character expression, phylogenetic reconstruction, and the detection of reticulate evolution, *Experimental and Molecular Approaches to Plant Biosystematics,* vol. 53, pp.333-354.

Robinson, D. F. & Foulds, L. R. (1981), Comparison of phylogenetic trees, *Mathematical Biosciences*, vol.53, no.1-2, pp.131-147.

Ropelewski, A. J., Nicholas, H. B. & Mendez, G. R. R. (2010), MPI-PHYLIP: parallelizing computationally intensive phylogenetic analysis routines for the analysis of large protein families, *PLoS One*, vol.5, no.11, pp.e13999.

Rzhetsky, A. & Nei, M. (1992), A simple method for estimating and testing minimum-evolution trees, *Molecular biology and evolution*, vol.9, no.5, pp.945-967.

Saitou, N. & Nei, M. (1987), The neighbor-joining method: a new method for reconstructing phylogenetic trees, *Molecular biology and evolution*, vol.4, no.4, pp.406-425.

Sawyer, S. (1989), Statistical tests for detecting gene conversion, *Molecular biology and evolution*, vol.6, no.5, pp.526-538.

Schaeli, B., Al-Shabibi, A. & Hersch, R. D. (2008), Visual Debugging of MPI Applications, In *Proceedings of the 15th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface*, pp.239-247.

Schmidt, H. A., Strimmer, K, Vingron, M. & von Haeseler, A. (2002), TREE-PUZZLE: maximum likelihood phylogenetic analysis using quartets and parallel computing, *Bioinformatics*, vol.18, no.3, pp.502-504.

Schmollinger, M., Nieselt, K., Kaufmann, M. & Morgenstern, B. (2004), DIALIGN P: fast pair-wise and multiple sequence alignment using parallel processors, *BMC Bioinformatics*, vol.5, no.1, pp.128.

Skillicorn, D. B. (1988), A taxonomy for computer architectures, *Computer*, vol.21, no.11, pp.46-57.

Sonea, S. & Panisset, M. (1976), Pour une nouvelle bactériologie, *Revue Canadienne de Biologie*, vol.35, pp.103-167.

Sneath, P. & Sokal, R. R. (1973), *Numerical taxonomy: The principles and practice of numerical classification*, WH Freeman San Francisco.

Sneath, P., Sackin, M. J. & Ambler, R. P. (1975), Detecting evolutionary incompatibilities from protein sequences, *Systematic Biology*, vol.24, no.3, pp.311-332.

Stamatakis, A. (2006), RAxML-VI-HPC: maximum likelihood-based phylogenetic analyses with thousands of taxa and mixed models, *Bioinformatics*, vol.22, no.21, pp.2688-2690.

Stephens, J. C. (1985), Statistical methods of DNA sequence analysis: detection of intragenic recombination or gene conversion, *Molecular Biology and Evolution*, vol.2, no.6, pp.539-556.

Stewart, C. A., Hart, D., Berry, D. K., Olsen, G. J., Wernert, E. A. & Fischer, W. (2001), Parallel Implementation and Performance of FastDNAml - A Program for Maximum Likelihood Phylogenetic Inference, In *Proceedings of the ACM/IEEE Supercomputing Conference*, pp.32.

Talbi, E. G. & Zomaya, A. Y. (2008), *Grid computing for bioinformatics and computational biology*, Wiley-Interscience.

Tan, G., Feng, S. & Sun, N. (2005), Parallel Multiple Sequences Alignment in SMP Cluster, In *Proceedings of the Eighth International Conference on High Performance Computing in Asia-Pacific Region*, pp.426.

Templeton, A. R., Crandall, K. A. & Sing, C. F. (1992), A cladistic analysis of phenotypic associations with haplotypes inferred from restriction endonuclease mapping and DNA sequence data. III. Cladogram estimation, *Genetics*, vol.132, no.2, pp. 619-633.

Than, C. & Nakhleh, L. (2008), SPR-based tree reconciliation: Non-binary trees and multiple solutions, In *Proceedings of the 6th Asia Pacific Bioinformatics Conference*, pp.251-260.

Thompson, J. D., Higgins, D. G. & Gibson, T. J. (1994), CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice, *Nucleic acids research*, vol.22, no.22, pp.4673.

Trapnell, C. & Salzberg, S. L. (2009), How to map billions of short reads onto genomes, *Nature Biotechnology*, vol.27, no.5, pp.455-457.

Trapnell, C. & Schatz, M. C. (2009), Optimizing Data Intensive GPGPU Computations for DNA Sequence Alignment, *Parallel computing*, vol.35, no.8, pp.429-440.

Treleaven, P. C. (1988), Parallel architecture overview, *Parallel Computing*, vol.8, no.1-3, pp.59-70.

Veen, A. & de Lange, M. (1993), Overview of the PREPARE Project, In *Proceedings of the Fourth Workshop on Compilers for Parallel Computers,* Delft, Holland.

von Haeseler, A. & Churchill, G. A. (1993), Network models for sequence evolution, *Journal of molecular evolution*, vol.37, no.1, pp.77-85.

Wall, L. (1994), *The Perl programming language*, Prentice Hall Software Series.

Walters, J. P., Meng, X. , Chaudhary, V., Oliver, T., Yeow, L. Y., Schmidt, B., Nathan, D. & Landman, J. (2007), Mpi-hmmer-boost distributed fpga acceleration, *The Journal of VLSI Signal Processing*, vol.48, no.3, pp.223-238.

Walters, J. P., Balu, V., Kompalli, S. & Chaudhary, V. (2009), Evaluating the use of GPUs in liver image segmentation and HMMER database searches, In *Proceedings of the International Symposium on Parallel and Distributed Processing Symposium* (IPDPS), pp.1-12.

Wilkinson, B. & Allen, M. (2004), *Parallel programming: techniques and applications using networked workstations and parallel computers*, Prentice-Hall, Inc.

Wolfe, M. & Banerjee, U. (1987), Data dependence and its application to parallel processing, In *Proceedings of the International Journal of Parallel Programming*, vol.16, no.2, pp.137-178.

Zomaya, A. Y. 2006, *Parallel computing for bioinformatics and computational biology*, John Wiley and Sons.

Zwickl, D. J. (2006), Genetic algorithm approaches for the phylogenetic analysis of large biological sequence datasets under the maximum likelihood criterion, *Ph.D. dissertation, University of Texas at Austin.*