Mémoire d'

HABILITATION À DIRIGER DES RECHERCHES

présenté à l'

UNIVERSITÉ D' EVRY VAL D'ESSONNE

Spécialité

INFORMATIQUE

Analyse Statique : du Calcul Haute Performance à la Bioinformatique

par :
FRANCK DELAPLACE

- devant le jury composé de -

M.	G.	BERNOT	EXAMINATEUR
M.	M.	CROCHEMORE	EXAMINATEUR
M.	J.	DEMONGEOT	RAPPORTEUR
M.	P.	FEAUTRIER	RAPPORTEUR
MME.	M.	KAUFMAN	EXAMINATRICE
M.	F.	KÉPÈS	EXAMINATEUR
M.	J.C.	KÖNIG	EXAMINATEUR
M.	J.	ROMAN	RAPPORTEUR

le 28 Novembre 2003

Table des matières

Chapitre 1

Introduction

La réalisation de logiciels scientifiques fait souvent apparaître des difficultés de conception qui ne sont pas seulement liées à la modélisation du problème traité mais aussi à l'implantation des problèmes correspondants (choix des structures de données, algorithmes, techniques logicielles liées aux architectures). Ces difficultés sont certainement partagées par de nombreux programmeurs face à n'importe quelle application, mais elles sont rendues cruciales dans les domaines scientifiques par la nécessité de *passer à l'échelle*, c'est-à-dire de *« croquer »* un grand nombre de données ou d'évaluer un grand nombre de solutions dans des temps raisonnables. Par exemple, la simulation météorologique et l'analyse de séquences d'un génome nécessitent des grandes puissances de calculs. Le premier exemple met plutôt en avant la nécessité d'effectuer des calculs sur des grandes masses de données alors que les difficultés du deuxième portent sur l'exploration de grands espaces de solutions. Afin de développer ces applications, on a souvent recours à l'utilisation de techniques algorithmiques sophistiquées et au portage de ces applications sur des architectures parallèles.

Se pose donc le problème d'acquérir la maîtrise du développement de techniques sophistiquées permettant de réaliser ces logiciels. La sophistication réclamée conduit assez naturellement à donner une place importante au développement.

Dans ce cadre, la recherche d'outils permettant une maîtrise de méthodes sophistiquées et des coûts de développement apparaît indispensable. L'esquisse de la solution que nous défendons peut être introduite par l'image suivante : l'ensemble des méthodes et des mécanismes impliqués dans la réalisation d'un logiciel se partage entre le programmeur et le langage. Ce que ne développe pas le programmeur en détail, le compilateur le réalise et inversement. Cette image est bien sûr schématique car il existe des tâches difficilement transposables d'un acteur du développement à l'autre. Par exemple, la programmation en binaire par un programmeur n'est pas envisageable. Toutefois cette image trouve en programmation parallèle des exemples où elle s'incarne bien. Nous en donnerons un exemple dans le chapitre 2. Elle met en valeur l'idée que le développement d'un logiciel peut se définir comme une quantité de travail que l'on peut répartir entre l'homme et l'agrégat langage/compilateur dans une mesure plus large qu'il n'y paraît.

En déplaçant le travail de développement du programmeur vers le langage/compilateur, on souhaite répondre aux problèmes posés par la programmation de ce type d'applications car le compilateur prendra mieux en charge les aspects purement techniques du domaine traité. Ce déplacement opère à la fois en nombre et en complexité : en nombre en réalisant le maximum de tâches routinières possibles et en complexité en réalisant les parties algorithmiques sophistiquées évoquées précédemment. Idéalement, la programmation se trouve alors résumée à quelques spécifications s'abstrayant

complètement de la machine, spécifications qui ne diffèrent guère de l'écriture mathématique du modèle.

L'intégration des techniques sophistiquées relevant de l'expertise informatique et des tâches routinières se déduit alors automatiquement de cette spécification. La programmation deviendrait une activité à portée des scientifiques non-spécialistes des métiers de l'informatique car la plupart des problèmes inhérents au passage à l'échelle y serait gommée.

Abandonnons provisoirement cette utopie pour nous concentrer sur ses modalités actuellement accessibles. D'un coté la programmation décrit les opérations dans un modèle de calcul correspondant à l'abstraction d'une machine. De l'autre, il existe une machine physique définissant le modèle d'exécution. Le déplacement précédemment mentionné s'incarne dans la traduction d'un programme *simple* dédié à un modèle de programmation en un autre *complexe* dédié au modèle d'exécution. Les règles régissant ce passage se déterminent par rapport à la sémantique du langage de programmation et à celle de l'exécution. Dans les chapitres de ce mémoire, nous développerons plus en détail des critères permettant d'établir avec plus d'objectivité la différence entre simplicité et complexité pour les langages.

Cependant l'expérience montre que ces règles présentent une *certaine inefficacité*. Tout du moins elles n'offrent pas toujours la performance à laquelle on peut s'attendre si l'on avait directement décrit le programme dans le modèle d'exécution. On impute cette inefficacité relative à la nécessité de couvrir tous les cas de traduction, obligeant ainsi à toujours définir cette traduction selon le schéma le plus complet et donc le plus complexe. La compilation des communications (chapitre 2) en constitue un exemple caractéristique.

En se fondant initialement sur une conception cohérente visant à simplifier la programmation, cette démarche se trouve réduite dans ses perspectives par la complication qu'impose la généralité de son traitement. Il faut donc avoir recours à des optimisations pour produire des alternatives performantes au cas général. Pour ce faire, les optimisations tirent bénéfice des particularités d'un programme pour le transformer d'une manière plus performante que ne l'aurait fait le processus standard de compilation. Il faut donc déduire à partir de la source du programme des propriétés spécifiques que le cas général ne possède pas. Sur quels critères distingue-t-on ces propriétés ? Sans pouvoir bien sûr les énumérer, on peut affirmer qu'ils se rapportent à la dynamique du programme, c'est-à-dire à son exécution.

Il faut donc identifier des catégories d'exécutions propices à définir des optimisations particulières. Cependant ce processus s'appliquera non pas sur une trace d'exécution mais sur le code source lui-même. Il s'agit donc d'une *analyse statique* dont l'objet est d'inférer des propriétés particulières à partir de la source du programme afin d'appliquer les optimisations conduisant à une exécution plus performante.

L'analyse statique laisse une place centrale à la définition de représentations chargées de donner une vision du programme qui soit adéquate à l'application d'optimisations. Par exemple, dans le cadre d'une parallélisation, on s'appuiera à la fois sur un graphe décrivant les dépendances entre les tâches et sur une modélisation géométrique des espaces de calculs par des polyèdres convexes (chapitre 3). La qualité des optimisations dépend de la pertinence des représentations vis-à-vis de propriétés spécifiques du programme à optimiser.

A cette présentation du cadre de nos recherches, il faut ajouter d'autres motivations que nous abordons à présent.

La conception de langage/compilateur dédié à un domaine contribue au développement de mo-

dèles et des modes de pensée de ce domaine. Ceci – non parce que la syntaxe est commune aux programmes considérés – mais parce que sa spécialisation dédie la sémantique du langage à un domaine. On espère ainsi *capturer une démarche du domaine concerné*, ce qui dans une certaine mesure le définit. C'est cette démarche que nous nommerons de manière raccourcie *le sens* de ce domaine.

Pour moi, vouloir rendre explicite cette démarche par la conception de langage est essentiel car on veut donner les moyens d'exprimer le sens que partagent les acteurs d'un domaine scientifique. Il s'agit de définir un cadre d'expression unifiant les discours dans un même creuset que constitue un langage. En informatique, on peut distinguer deux orientations du terme expression : le pouvoir et le style.

Le pouvoir d'expression caractérise la capacité à décrire la solution d'un problème dans un langage interprété par une machine. Les limites de ce pouvoir se rapportent donc à celles du modèle de calcul de la machine. Tout langage dont l'exécution se fait sur un ordinateur ne possède donc pas un pouvoir d'expression supérieur à celui d'une machine de Türing. Si cette réflexion est poussée à son extrême, un unique langage suffit.

Tout programmeur sait cependant qu'une autre composante concernant le style de programmation possède une grande importance . En informatique, le style se nomme expressivité. Cette démarche méthodologique de conception des langages a pour objectif de simplifier l'expression en proposant de nouvelles constructions capables de capturer ce sens. La conception d'un langage/compilateur dédié à un domaine veut donc apporter un *supplément d'expressivité et non pas un pouvoir d'expression supplémentaire*. Elle se fonde sur le constat que les langages/compilateurs généralistes ne tirent pas parti de l'unité que l'on perçoit à la lecture des spécifications ou des sources des logiciels d'un domaine. Notons que cette unité ne se réfère pas nécessairement à une abstraction. Elle s'incarne très bien par la fréquence d'apparition de certaines structures de programmes, ou plus simplement de certaines fonctions. Une fois cette unité cernée, on peut donc espérer mieux décrire ou mieux exécuter un programme en donnant une place centrale à celle-ci dans la conception du langage et du compilateur.

Il s'agit là d'un pari qui affirme en miroir à la citation de Boileau [Boi74] *« ce qui se conçoit bien s'énonce clairement et les mots pour le dire viennent aisément »* que *« ce qui s'énonce bien se conçoit clairement et les concepts pour penser viennent aisément »*. D'une manière pragmatique, l'aisance fait référence à *l'alliance de deux principes* :

- Premièrement d'incarner les concepts d'un domaine de sorte que la distance entre le vocabulaire de l'utilisateur et celui du langage se réduise. Cette approche se destine plus particulièrement à la conception des langages dédiés à un domaine tel que TAGCC pour la recherche de gènes (chapitre 5). Si l'on élargit celle-ci à une activité de programmation nouvelle, on s'aperçoit qu'elle guide aussi la conception de langages destinés à de nouvelles manière de calculer comme celle de calculer en parallèle (chapitre 4). Ce principe souligne essentiellement l'importance à accorder à la capture des concepts d'un domaine en se tournant vers l'utilisateur et vers les applications.
- Deuxièmement de fournir des abstractions suffisantes pour masquer des problèmes plus directement posés par l'architecture. Cette approche repose en grande partie sur la capacité du compilateur à opérer la traduction des abstractions proposées en un exécutable. Ce principe conduit à concevoir des mécanismes capables *« d'absorber »* et parfois de niveler l'évolution des architectures du point de vue de leur utilisation (chapitre 2,3,4,5).

Dans cet état d'esprit, la recherche de performances peut apparaître secondaire car l'expressivité à elle seule semble justifier cette démarche. En fait il n'en est rien ; simplement son rôle change. Elle

sert selon moi de guide objectif – car quantitatif – pour déterminer les mécanismes représentant le mieux cette capture de sens par des méthodes informatiques. Dans une certaine mesure elle permet d'établir, en tenant aussi compte de l'expressivité, une classification entre différents mécanismes. Se pose alors le problème de trouver les articulations nécessaires pour que la concision et la clarté d'un programme source soit doublé de la performance du traitement. Il s'agit de concevoir des méthodes pour *une expressivité raisonnée* qui intègre le besoin de performance, garant de sa pertinence.

Du massivement parallèle au parallélisme de données (chapitre 2)

Popularisé dans les années 80 par le livre de Daniel Hillis la *« Connection Machine »* [Hil85], le parallélisme massif se dévoilait selon son auteur comme le paradigme manquant qui permettrait aux machines d'en fin accéder à l'Intelligence. Plus par le nombre d'unités de calcul fortement interconnectées que par la performance effective obtenue, les architectures massivement parallèles nous donneraient enfin les moyens d'atteindre une échelle de calcul dont seul le cerveau était détenteur. Myriade d'unités calculantes, connectiques fortes, décentralisation du contrôle, coopération inter-unités, accomplissement en parallèle d'un même macro-calcul, tout concourait à échafauder – par une mimétique fondatrice – des modèles de programmation et d'exécution sur lesquels s'appuieraient le développement des programmes de l'Intelligence. Les architectures massivement parallèles se faisaient le substrat de l'intelligence artificielle.

Le parallélisme massif s'ouvrait aussi à de nouvelles applications. Les problèmes numériques posés par la physique, par l'imagerie, par la climatologie ... se profilaient à présent comme autant d'applications du parallélisme massif, repoussant de plus en plus haut le barreau à passer sur l'échelle des Mégaflop. La CM2 succéda à la CM1, elle-même supplantée par la CM200, puis la CM5. Les générations successives des Connection Machines traçaient l'évolution du parallélisme massif vers une quête de performances croissantes que réclamaient ces nouvelles applications. Les applications du *Grand Challenge* désignaient le Téraflops comme l'ultime barreau ; Himalaya du calcul massivement parallèle dont le défi que représentait sa conquête se faisait source d'inspiration.

L'explosion des constructeurs d'architectures parallèles présentant un florilège de modèles durant les années 90 allaient bientôt laisser place aux grappes de PCs [ea95] comme modèle dominant pour les architectures parallèles. Le tamis des compromis technologiques et économiques avait donc œuvré pour sélectionner comme standard d'architectures parallèles des PCs grand public interconnectés par un réseau à haut débit. Un avantage que présente cette standardisation est d'imposer clairement un modèle d'exécution à plusieurs flots de contrôle indépendants et à mémoire distribuée (modèle MIMD) comme le paradigme du modèle d'exécution des architectures parallèles. Le langage se rapportant à ce modèle provient de l'association d'un langage séquentiel (C ou FORTRAN) et d'une bibliothèque de communications (MPI ou PVM).

Le parallélisme massif se fondait sur l'emploi d'un très grand nombre de processeurs. Et très tôt s'est posée la question d'établir le profil des applications qui pouvaient consommer un tel débit d'opérations. Il apparaissait clair que le parallélisme de tâches distinguant pour chaque processus un traitement différent n'y suffirait pas car personne ne peut concevoir un millier de tâches différentes en parallèle. Inévitablement, s'imposait donc le parallélisme de données où un traitement unique s'applique en parallèle sur un grand nombre de données. Les langages du parallélisme de données développent ainsi en marge du modèle MIMD un modèle de programmation propre que l'on peut assimiler à un modèle à flot de contrôle unique et à mémoire centralisée (modèle SIMD). Dans la mesure où les deux modèles s'imposaient de manière cohérente – mais par des motivations diffé-

rentes – comme référence pour la conception d'architectures et de langages, il n'y avait pas lieu de pointer la difficulté à vouloir traduire l'un dans l'autre, mais de s'atteler à résoudre le verrou de cette traduction jugé alors essentiel :

Comment émuler un espace d'adressage unique sur une architecture à mémoire distribuée ?

L'expressivité des langages parallèles repose précisément sur la capacité à cacher le parallélisme en le rendant implicite, démarche fidèle au second principe précédemment évoqué qui consiste à fournir des abstractions suffisantes pour masquer les problèmes d'implantation. Dans ce cadre, l'un des facteurs fondamental d'expressivité des langages à parallélisme de données est de dissimuler l'éclatement de l'espace d'adressage mémoire des grappes de PCs en reproduisant un espace d'adressage unique, comme c'est le cas dans les langages séquentiels.

Il s'agit donc de déterminer les techniques de compilation qui permettent de traduire efficacement les programmes décrivant des accès à une mémoire centralisée en un programme dont les échanges de données nécessitent de communiquer les données par passage de messages. L'ensemble de ces techniques définissent la *compilation des communications*.

Quelques irrégularités faites de manière régulière (chapitres 3 & 4)

Mettant à profit certaines particularités ou régularités d'un programme, un compilateur optimisant améliore le code généré pour accélérer le traitement. Ces particularités sont relevées à partir d'une source d'un programme en l'analysant statiquement. Ces analyses infèrent des propriétés sur l'exécution afin de déterminer une alternative à l'exécutable produit en suivant les règles de la sémantique standard du langage. Elles sont cependant délimitées par un cadre applicatif qui correspond à un sous-ensemble des constructions syntaxiques possibles que l'on peut écrire dans un langage. Hors de ce cadre, elles deviennent inopérantes. Les optimisations comme la parallélisation automatique ou celles se rapportant à la compilation des communications possèdent comme cadre d'analyse les programmes à *contrôle statique* [Fea91] ou programmes «réguliers». Cette analyse se base sur une modélisation du calcul (de l'exécution) par un formalisme géométrique (polyèdres convexes). Une question naturelle est alors de tenter d'étendre le cadre d'analyse de programmes à contrôle statique.

Une catégorie importante d'applications n'entrant pas dans ce cadre correspondent à des applications qualifiées *d'irrégulières*. Comme l'indique ce terme, elles exhibent une irrégularité qui rend complexe les optimisations tant l'exécution apparaît imprévisible. Toutefois en s'intéressant à l'origine de cette irrégularité, il est possible de trouver un *« biais »* qui permette d'écrire ce type de programmes dans le cadre du contrôle statique. Ainsi, on bénéficie de la puissance des outils du calcul polyédrique. Cependant, l'interprétation que l'on en fait diffère car le programme s'exécute différemment de son homologue régulier.

L'une des origines de l'irrégularité provient de certaines structures de données dont la gestion rend le programme irrégulier. Les matrices creuses correspondent à ce cas. Il s'agit de structures de données correspondant initialement à des tableaux mais dont le grand nombre de 0 stockés et leur très grande taille conduit à compresser un tableau en éliminant les 0. Toutefois pour des raisons évoquées au chapitre 3, l'accès se gère comme un tableau. L'antagonisme entre la représentation physique compressée et l'usage que l'on en fait engendre l'irrégularité du traitement (cf. chapitre 3).

En déplaçant le choix de la structure creuse du programmeur au compilateur, on élimine l'irrégularité du programme car le programmeur a l'illusion de traiter un tableau classique. Charge au

compilateur de gérer la complexité de l'irrégulier inhérent aux structure creuses. Les programmes entrent de nouveau dans le cadre syntaxique fixé par le contrôle statique et on peut représenter leur exécution par des polyèdres convexes. *L'irrégulier se traite alors avec les mêmes outils que le régulier mais en utilisant différemment l'analyse faite par ces outils.*

Un des éléments intéressant d'un calcul irrégulier opérant sur une structure creuse est d'exhiber un parallélisme supplémentaire que son homologue régulier ne possède pas. Dans l'algorithme parallèle de factorisation de Cholesky pour des structures creuses [HNP91], une parallélisation particulière du programme existe qui permet d'exécuter en parallèle des tâches opérant sur les colonnes. Bien que sa version opérant sur des structures *« denses »* peut être parallélisée, les méthodes automatiques de parallélisation ne découvrent pas le parallélisme permettant l'exécution parallèle de taches opérant chacune sur une colonne.

Clairement, cela signifie que *ce parallélisme échappe aux méthodes classiques de parallélisation automatique* opérant sur les programmes à contrôle statique. Il existe donc une propriété particulière se rapportant au 0 que l'on utilise pour paralléliser l'algorithme. Se pose donc la question :

Comment formaliser le parallélisme du creux pour paralléliser automatiquement ?

La réponse à cette question m'a conduit à la conception d'une nouvelle approche de l'analyse que nous avons qualifiée de *semi-statique*. Il s'agit de combiner des traitements d'analyse effectués à la compilation avec des traitements d'analyse effectués à l'exécution (run-time analysis) pour exploiter le parallélisme du creux. Le chapitre 3 expose cette formalisation.

Cette étude menée en amont du processus de compilation appelle en aval une étude pour la conception d'un *« back-end »* de compilation générant un code parallèle creux *pour valider complètement cette méthode*. Nous avons prévu que cette partie s'appuie sur une librairie pour les structures creuses qui soit adaptée à la génération d'un code creux dans les conditions de la parallélisation creuse. En considérant les contraintes de la parallélisation, l'objet de cette étude est de

> *proposer une librairie objet* STL *définissant une structure creuse « universelle » fondée sur le parallélisme de données qui masque le creux et dont l'exécution s'effectuerait sur une grappe de PCs.*

Ainsi exprimée, la spécification de cette librairie nommée PARADEIS vise à établir les compromis pour obtenir une librairie préservant l'essentiel des fonctionnalités dans les trois axes mentionnés (creux, parallélisme de données, grappe de PCs).

L'application de cette librairie dépasse en réalité celui de la génération de code car l'expressivité qu'elle apporte s'adresse aussi à un programmeur. En effet, elle affranchit ce dernier des contraintes de la manipulation des structures creuses en parallèles. Le chapitre 4 détaille ces travaux.

Le programme génome et le programmeur (chapitre 5)

L'informatique peut-elle contribuer à l'analyse du génome autrement qu'en proposant des logiciels ? A la première lecture, cette question apparaît curieuse tant la réponse par la négative semble évidente. Pourtant, une autre réponse peut être apportée en levant l'ambiguïté qui existe entre les outils informatiques et l'informatique comme discipline scientifique. Pour un scientifique non informaticien, cette distinction n'a que peu d'importance car l'informatique fournit essentiellement les outils pour obtenir des réponses aux questions de son domaine de recherche. Toutefois, c'est précisément dans la perspective de fournir de nouveaux outils que cette question mérite d'être posée pour l'étude du génome. Examinons les deux aspects :

Donald Knuth affirmait que *« la biologie a au moins* 500 *ans de problèmes très intéressants à poser aux informaticiens »*. En même temps qu'un appel à s'attaquer à des problèmes dont la difficulté séduit, cette citation jette les bases *d'un certain rapport* entre la biologie et l'informatique. La biologie propose des problèmes qu'une modélisation transforme en problèmes algorithmiques. En retour les solutions algorithmiques conduisent, par le développement de logiciels, à fournir des outils qui permettront d'avancer dans l'étude des questions biologiques. L'informatique attend donc les problèmes de la biologie pour y répondre de manière adaptée par un logiciel. Cette démarche présente indiscutablement de l'intérêt. Un regard sur les articles de bioinformatique et sur les logiciels disponibles sur Internet suffit à s'en convaincre (site [Ana]). Mais est-ce la seule démarche présentant un intérêt pour la biologie et pour l'informatique ?

Pour introduire une autre possibilité, nous nous appuierons sur une autre citation, celle du biologiste Henri Atlan dans son livre *« A tort et à Raison »* [Atl86] pp. 57 : *« Plus récemment la biologie nouvelle a découvert et posé l'organisation de la matière comme l'objet véritable de sa discipline de deux façons. D'abord en ne pouvant faire autrement que d'emprunter aux sciences de l'information et des organisations artificielles (machines programmables, automates) les concepts fondamentaux qui lui ont permis ces avancées théoriques... »*. En biologie moléculaire les concepts de code et de programmes génétiques et la représentation des régulations des gènes par des graphes suffisent à accréditer cette citation. Ainsi, la biologie utilise des concepts *que l'informatique développe pour elle-même* pour comprendre ses propres problèmes. La contribution à la biologie provient alors de la conception d'outils théoriques et appliqués développés *en interne* à la discipline informatique sans attendre les problèmes biologiques.

Cette démarche repose sur une *similitude des schémas conceptuels* utilisés pour forger les réponses aux problèmes posés par la biologie et par l'informatique. Loin de s'opposer à la première, elle la conforte en ouvrant d'autres perspectives qui répondent *de manière indirecte* aux problèmes posés par la biologie. Cette intuition m'a conduit à concevoir le langage TAGCC pour l'annotation des génomes.

Une fois le séquençage d'un génome terminé, on dispose de séquences formées de quatre symboles A,T,G,C représentant les macro-molécules d'ADN qui constituent les chromosomes. Schématiquement, l'annotation consiste à rechercher des régions d'intérêt biologique dans ces séquences numérisées. La région recherchée la plus essentielle est bien sûr celle correspondant à un gène, région qui code pour une protéine. Pour des raisons que nous expliquerons dans le chapitre 5, il n'existe pas une méthode unique de recherche de gènes car il n'existe pas une modélisation exacte de la structure d'un gène.

Dans ce cadre, on est amené à concevoir différentes méthodes, à les tester, à les combiner, à les faire évoluer, à les adapter à différents génomes, à les spécialiser à une partie de l'analyse, à en chercher enfin de nouvelles. L'espace des problèmes intéressants apparaît ici d'une ampleur que ne renierait pas Donald Knuth. Toutefois, pour l'avoir expérimenté, le coût de développement est à la hauteur de cette vastitude. De surcroît, durant les aller-retours entre la validation et la conception, il apparaît important que les comparaisons entre méthodes soient formalisées de sorte que l'on puisse extraire facilement des éléments pour renforcer la modélisation.

Pour des raisons qui ont été développées précédemment et d'autres qui seront plus spécifiquement étudiées pour l'annotation dans le chapitre 5, un langage dédié à un domaine permet d'accélérer le passage de la conception à la programmation. De plus, la spécialisation du langage réduit la distance entre le programme et le modèle. Ainsi, l'analyse des programmes, automatique ou non, donne des informations sur les modèles qui seront plus faciles à obtenir qu'à partir d'un programme rédigé dans un langage généraliste. La conception d'un langage dédié à l'annotation doit être expressif

mais aussi, au vu de la taille des données traitées, être efficace. Un tel langage devra donc apporter une réponse à la question :

Comment concevoir un langage dédié à l'annotation des génomes qui concilie à la fois expressivité et performance ?

Vers la modélisation de la dynamique des réseaux biologiques (chapitre 6)

La biologie moderne s'est donnée comme objectif d'expliquer les systèmes biologiques par des phénomènes biochimiques à l'échelle moléculaire. Cette démarche conduit à un double processus de désassemblage des systèmes jusqu'à leur composants moléculaires et d'assemblage par construction des systèmes biologiques au sein desquels ces constituants exercent une fonction.

Bien qu'essentiel, l'inventaire des gènes d'un génome ne suffit pas à cerner le comportement d'un organisme car celui-ci se rapporte aussi aux interactions protéiques et géniques. Son étude implique d'élargir le champ d'analyse d'un génome à son environnement moléculaire afin de prendre en compte les interactions qui gouvernent la dynamique d'expression des gènes. Cet élargissement s'appuie sur l'analyse des données du génome, du transcriptome et du protéome.

Cependant, cette analyse révèle tout d'abord la complexité des systèmes biologiques, plus la difficulté à les interpréter en regard des fonctions cellulaires que par leur volume et la variété des interactions. La complexité se présente alors comme l'un des objets centraux d'étude de la biologie moderne dont la compréhension consiste à établir le lien entre le comportement d'un organisme et les interactions moléculaires. Dans ce cadre, l'arrivée massive des données biologiques *pose de manière centrale une question de méthode* pour l'étude de la complexité qu'elles révèlent.

Le passage à l'échelle et la complexité des paramètres impliquent de développer des méthodes adaptées qui se fondent sur des approches incrémentales et modulaires. L'étude de la modularité ne se justifie cependant pas uniquement pour des raisons de performances. Elle est considérée comme un principe méthodologique d'explication de la complexité de la régulation car elle permet de faire émerger une structuration du réseau allant des composants les plus élémentaires aux éléments les plus intégrés.

Pour décrire leurs interactions, les graphes offrent une abstraction permettant de décrire des composants biologiques de nature différente de manière homogène. Dans ce cadre, Les graphes capturent une partie de la complexité biologique dans des abstractions calculables.

Les interactions entre gènes se présentent alors sous une forme qui pose les bases d' une modélisation de la régulation utilisant des formalismes appliqués en informatique.

Dans le chapitre 6, nous arborderons les résulats que nous avons obtenus sur la modélisation des réseaux de régulation génétiques, et la manière dont nous avons adapté certaines méthodes d'analyse statique pour structurer les réseaux génétiques en module. De ces travaux, nous esquisserons une démarche prospective pour la suite de ces recherches.

Structure des chapitres

Chaque chapitre décrit en premier lieu les motivations des axes de recherches menés. Puis nous décrirons schématiquement les points d'articulation principaux menant aux résultats. Enfin, nous situerons les recherches menées avant d'établir un bilan. Le dernier chapitre décrivant une recherche encore en cours, nous donnerons les perspectives ouvertes par les travaux que nous avons déjà réalisés.

Chapitre 2

Compilation de communications prédictibles

2.1 Introduction

L'un des objectifs présidant à la conception de langages parallèles de haut niveau concerne la recherche d'abstraction pour simplifier la programmation parallèle. Les critères de simplicité pour la programmation de programmes parallèles font appel à des critères qualitatifs qui varient nécessairement selon les utilisateurs. Toutefois, il est possible d'établir une classification en prenant comme principe que la programmation la plus simple est celle qui ressemble le plus à celle d'un langage séquentiel et, par opposition, la programmation la plus complexe est celle qui décrit explicitement le parallélisme et les interactions avec des processus s'exécutant en parallèle. Ce principe incarne de manière pragmatique l'abstraction d'un langage parallèle vis-à-vis de mécanismes d'utilisation du parallélisme.

Ainsi, on peut retenir deux critères permettant d'estimer la complexité de programmation :

- Le premier critère met en relation l'activité des processus et la mémoire. Elle oppose la notion d'espace d'adressage unique où tous les processus accèdent à une mémoire unique et l'espace d'adressage multiple où chaque processus possède un espace de mémoire propre.
- Le second critère se détermine par rapport aux structures de contrôle parallèle. Ces structures, quand elles existent, permettent d'exprimer la mise en concurrence d'opérations.

Il est possible de déterminer une échelle de difficulté de programmation déterminant le travail que doit effectuer l'utilisateur pour paralléliser un programme séquentiel. Les tâches à accomplir sont les suivantes :

1. déterminer le placement des données sur chacun des processeurs.

2. déterminer les coopérations entre tâches (synchronisation).
3. déterminer la conversion des adresses mémoire pour les accès à la mémoire globale en accès aux mémoires distribuées.

En considérant les langages impératifs, nous nous proposons d'illustrer les différentes approches par l'expression du programme séquentiel effectuant la somme de deux vecteurs avec décalage.

La figure 2.1 décrit ce même programme dans 3 langages parallèles différents : HPF [HPF], OPENMP [Ope] et MPI [For95]. Brièvement, les principales caractéristiques de ces langages sont les suivantes : En HPF, le type de parallélisme est extrait de la sémantique des opérations. Il s'agit d'un parallélisme de données. C'est l'interprétation des structure de contrôle (par ex. `forall`) qui produit l'exécution parallèle. En OPENMP, l'exécution parallèle s'exprime en fonction de directives de compilation appliquées en amont des boucles déclarant que l'exécution de celles-ci peut être traitée en parallèle. Chaque tâche traite un nombre d'itérations de la boucle en parallèle. La dernière approche que nous présentons est celle concernant l'emploi des bibliothèques de communication (MPI). Les processus s'exécutent indépendamment des uns des autres et accèdent uniquement à leur propre espace mémoire. Chaque processus peut exécuter un programme différent plutôt qu'une partie des itérations d'une boucle comme montré précédemment. De même, toute donnée nécessaire à un calcul qui n'est pas présente sur le processeur doit explicitement être échangée par communication au travers d'un protocole entre l'émetteur et le récepteur (qui doit de plus gérer le stockage local). Cet échange dépend lui-même du placement des données.

Dans les deux premiers langages, le parallélisme s'exprime par l'intermédiaire d'une *« boucle parallèle »* dont une présentation plus détaillée sera donnée à la section suivante (cf. figure 2.4). Dans le dernier, l'exécution parallèle correspond à celle de plusieurs processus communicant pouvant exécuter le même programme (programmation SPMD).

Un critère objectif de l'expressivité, parmi d'autres, peut être donné par le nombre de lignes d'un programme. Si celui-ci n'exprime pas la clarté, il mesure la concision. La comparaison des programmes présentés en figure 2.1 ainsi que le tableau 2.1 mettent clairement en évidence que fournir un espace d'adressage unique et des opérations globales permettant une expression centralisée du parallélisme constitue des éléments de simplification importants de la programmation parallèle.

Ainsi dans HPF, l'expression du parallélisme se détermine de manière abstraite par l'utilisation de la boucle `forall`. La mémoire est une mémoire d'adressage unique. La sémantique de cette structure de contrôle et la représentation de la mémoire détermine une machine SIMD à mémoire centralisée qui constitue un paradigme de programmation parallèle facile à utiliser. A l'autre bout de la chaîne, la conception des architectures physiques répond à d'autres motivations qui privilégie les grappes de PCs dont le modèle d'exécution est un modèle MIMD à mémoire distribuée où les processus communiquent par un protocole d'envoi et de réception de messages. En terme de programmation, le langage le plus proche de ce type d'architecture est celui d'un langage séquentiel auquel s'ajoute une librairie de passage de messages (MPI). Dans ce cadre, compiler des communications revient à transformer automatiquement des accès à une mémoire unique centralisée en instructions de communications.

Précisons que des paradigmes différents de la programmation impérative existent pour la programmation parallèle. Les langages à flots de données offrent aussi une formulation intéressante et simple de la gestion parallèle des calculs. Proposé par G. Kahn en 1974 comme un schéma général de programmation, le langage KPN (Kahn Process Networks) [Kah74] définit les bases théoriques de cette classe de langages parallèles. Le modèle de calcul correspond à des processus parallèles communicants par des files FIFO de tailles infinies.

	Distribution	*Coopération*	*Conversion*
HPF	utilisateur	compilateur	compilateur
OPENMP	utilisateur	compilateur	compilateur
MPI	utilisateur	utilisateur	utilisateur

TAB. 2.1 – Répartition du travail entre l'utilisateur et le compilateur

Pour ces langages, la gestion de la mémoire est différente. Cette expression offre aussi un cadre simple de programmation. Nous renvoyons le lecteur aux articles de R. stephens [Ste97] et de J.L. Giavitto [Gia99] pour un résumé complet des langages fondés sur la notion de flots (*stream processing*). Bien qu'ils s'agissent de langages généralistes, le cadre applicatif de ces langages concernent plutôt les domaines des systèmes embarqués, de la conception de circuits et du traitement du signal.

Pour nos travaux, nous avons choisi de nous concentrer sur les langages utilisés pour programmer les architectures multiprocesseurs que sont les langages parallèles impératifs.

Les solutions proposées que nous avons développées en collaboration avec Cécile Germain [GD97, GD95, GDC94] ont pour objectif de déterminer un schéma de compilation des communications qui maximise la performance d'exécution du programme produit. Cette maximisation se détermine par rapport à un modèle estimant le coût en temps des communications. En considérant un *modèle linéaire* le temps d'envoi de m messages de taille n, $T_c(n, m)$, est le suivant :

$$T_c(n, m) = m.(T_I + T_D.n)$$

où T_I est le temps d'initialisation du message. Ce coût fixe est imputé au protocole de communication nécessaire à la délivrance d'un message. T_D est le coût imputé au transfert d'une donnée. Il y a 3 ordres de grandeurs entre ces deux constantes ($10^3.T_D \approx T_I$). Il impose donc de minimiser la grandeur $m.T_I$ en premier lieu. L'alternative possible est :

- *Minimiser* T_I : il s'agit de dimininuer le coût fixe de chaque message. Cette réduction s'effectue en supprimant des traitements logiciels impliqués dans le routage des messages ; c'est-à-dire de réduire le protocole au transfert des données. Dans l'architecture PTAH, proposée par Franck Cappello [Cap94] et pour laquelle j'ai conçu et réalisé le compilateur, l'unique information de routage *à l'exécution* était une *« carte »* (ou schéma) du routage qui contenait toutes les informations nécessaires à l'acheminement d'un message.
  En particulier, pour chaque phase de communications comportant plusieurs communications élémentaires point-à-point, le chemin de toutes les communications et la taille des tampons à réserver étaient déterminés. La carte devait décrire une communication sans conflit. La réduction du coût provenait du fait que *le calcul des cartes s'effectuait à la compilation* ; supprimant ainsi ce traitement de l'exécution.
  A la compilation, chaque échange de données était découpé en phase de communications où pour chaque phase une carte décrivait le routage à opérer. Cette dernière était placée dans une mémoire accessible par les routeurs. Ainsi une communication, correspondant à une phase donnée, réduisait le protocole à une lecture d'une carte dans la mémoire des routeurs. Ces derniers établissaient alors les canaux nécessaires à cette communication. Le programme se découpait en une alternance de communications globales et de calculs parallèles. Les cartes de routage étaient automatiquement calculées à partir d'un programme écrit dans un langage à parallélisme de données – nommé HPT – qui correspondait à un sous ensemble de HPF. Ce

– Séquentiel –

```
#define N 1000
main(int argc,char** argv) {
  float V1[N],V2[N],R[N];
  int i;
  for(i=1;i<N-1;i++)
    R[i]= V1[i+1]+V2[i-1]; }
```

– HPF –

```
PARAMETER(N = 1000)
PARAMETER(NP=NUMBER_OF_PROCESSORS())
FLOAT R(N),V1(N),V2(N)
!HPF$ TEMPLATE T(N)
!HPF$ PROCESSORS P(NP)
!HPF$ ALIGN R WITH  T
!HPF$ ALIGN V1 WITH T
!HPF$ ALIGN V2 WITH T
!HPF$ DISTRIBUTE T(BLOCK) ONTO P(NP)

      FORALL(i=2:N-1)
        R(i)= V1(i+1)+V2(i-1);
      ENDFORALL
```

– OpenMP –

```
#define N 1000
main(int argc,char** argv) {
  float V1[N],V2[N],R[N];
  int i;
#pragma omp \
        parallel for\
        private(i)

  for(i=1;i<N-1;i++)
    R[i]= V1[i+1]+V2[i-1]; }
```

– MPI –

```
#include <malloc.h>
#include <mpi.h>
#define N 1000
#define COMM_V1 10
#define COMM_V2 20

main(int argc, char **argv) {
  int numprocs, myid;
  int size,i;
  float *V1,*V2,*R,v1,v2;
  MPI_Status dummy;
  MPI_Init(&argc,&argv);
  MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
  MPI_Comm_rank(MPI_COMM_WORLD,&myid);

  /* DISTRIBUTION PAR BLOC (version simple)  */
  if (N%numprocs == 0) {size=N/numprocs;} else {exit(0);}

  if((V1=(float*)malloc(size*sizeof(float)))==NULL)
    exit(0);
  if((V2=(float*)malloc(size*sizeof(float)))==NULL)
    exit(0);
  if((R =(float*)malloc(size*sizeof(float)))==NULL)
   exit(0);
  /* + Affectation de V1 et de V2 non écrites */

  /* COMMUNICATION + CALCUL */
  if (myid != 0) {
      MPI_Send(&V1[0],1,MPI_FLOAT,myid-1,
               COMM_V1,MPI_COMM_WORLD);
      MPI_Recv(&v2,1,MPI_FLOAT,myid-1,COMM_V2,
               MPI_COMM_WORLD,&dummy);
      R[0]=V1[1]+v2;}

  if (myid != numprocs-1) {
      MPI_Send(&V2[size-1],1,MPI_FLOAT,myid+1,
               COMM_V2,MPI_COMM_WORLD);
      MPI_Recv(&v1,1,MPI_FLOAT,myid+1,
               COMM_V1,MPI_COMM_WORLD,&dummy);
      R[size-1]=v1+V2[size-1];}

  for (i=1;i<size-1;i++) R[i]=V1[i+1]+V2[i-1];

  MPI_Finalize();}
```

FIG. 2.1 – Trois programmes parallèles, deux styles

mode de routage fut nommé *« routage statique »*.

- *Minimiser m* : cette approche repose uniquement sur le traitement effectué par le compilateur. Il s'agit d'un traitement de compilation qui ordonne les échanges de données à réaliser de sorte à grouper le plus possible les données dans un même message pour réduire le nombre de messages transmis.
  Ceci tend à faire croître n et à diminuer m. Le rapport entre T_I et T_D conduit à réduire le temps total de communications. Le processus de minimisation dépend du programme en lui-même. La compilation des communications consiste à déterminer quelles communications peuvent être groupées par l'analyse du programme source.
  Ce mode de communication est considéré comme une optimisation de celui de traduction générique des accès mémoire en programme de communications (ensemble d'instructions d'envoi et de réception). Il constitue une optimisation appelée *« vectorisation des communications »*. Celle-ci est prépondérante pour accélérer le temps de communications [Tse93]. Cependant, elle ne s'applique qu'à un sous-ensemble des constructions du langage ; sous-ensemble appelé les programmes à contrôle statique par Paul Feautrier [Fea91, Fea92a, Fea92b].

Considérons la première approche. L'une des conditions essentielles de sa faisabilité repose sur la capacité à compiler ces cartes. Dans ce cadre, il apparaît indispensable de ne pas produire autant de cartes qu'il existe d'opérations[1], c'est-à-dire de pas d'exécutions, car cela reviendrait à évaluer le programme à la compilation. Formulée autrement, cette condition implique que le nombre des cartes de communications produites soit de l'ordre du nombre d'instructions dans le programme et non pas du nombre d'opérations afin de l'intégrer dans un cadre de compilation opérant uniquement sur les instructions. Si l'on considère à présent une boucle séquentielle, on doit clairement déterminer un moyen d'engendrer un nombre de cartes qui ne dépend pas du nombre d'itérations de la boucle. Dans ce cadre, il apparaît nécessaire d'associer une unique carte à plusieurs communications, *et ainsi vectoriser les communications.*

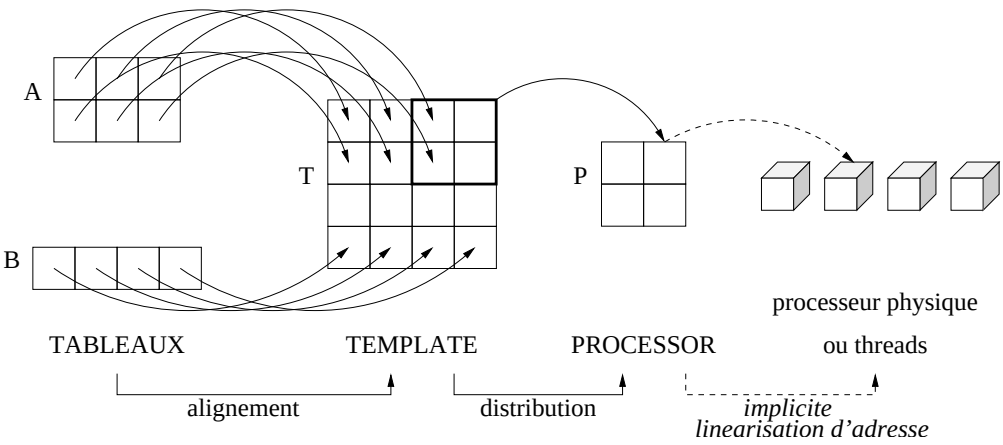
Puisque l'alternative proposée pour réduire le coût $m.T_I$ possède des exigences identiques en terme de compilation, elle nous a conduit à développer une solution commune dont l'objet est de produire un programme de communications vectorisées.

Après avoir décrit plus en détail les boucles parallèles et les directives du placement des données (section 2.2), nous déterminerons dans la section 2.3 les principales étapes de la compilation des communications. Puis la section 2.4 expose l'état de l'art dans ce domaine (section 2.5).

2.2 Langages à parallélisme de données

Pour expliquer en détail les solutions proposées, rappelons d'abord les éléments de syntaxe des langages à parallélisme de données qui entrent en compte pour la génération de code de communications. Cette section s'attache essentiellement à décrire les deux notions que sont les directives de placement et les boucles parallèles. Pour plus d'informations nous invitons le lecteur à consulter les ouvrages [GUD95, Paz97] pour des ouvrages français et [Fos95, Wol96] pour des ouvrages anglo-saxons.

[1] Rappelons qu'une opération correspond à une instruction en exécution.



En HPF le placement s'effectue en deux étapes. La première consiste à *aligner* les données sur un *template* qui schématiquement incarne une structure de processeurs *virtuels* dont le nombre n'est pas bornés par celui d'une machine physique. Les tableaux d'une application sont donc placés sur ce template de sorte que les positions relatives des tableaux les uns par rapport aux autres soient identifiées. Puis, ce template est *distribué* sur un tableau de processeurs de même dimension que le template. Le placement sur une architecture physique est à la charge du compilateur.

FIG. 2.2 – Règles de placement HPF

2.2.1 Directives de Placement des données

Les directives de placement de données décrivent la manière dont les tableaux sont distribués sur chacun des processeurs de la machine. Ce placement sera décrit pour le langage HPF. Des directives similaires se retrouvent aussi dans les langages ayant inspiré la conception de HPF comme FORTRAN D [HKK+91] et Vienna fortran [CMZ92]. Ces directives s'appuient sur une représentation de l'architecture physique ou logicielle comme *un tableau de processeurs*. Les adresses des processeurs se déterminent comme les adresses des cellules d'un tableau. Ainsi, le placement revient à plonger les tableaux de données dans le tableau de processeurs selon une loi donnée par les directives (figure 2.2). Le placement en HPF est décomposé en deux étapes : *l'alignement* et la *distribution*. L'alignement place les tableaux dans un tableau virtuel de processeurs nommé *template* qui sera lui-même plongé dans un tableau de processeurs plus petit nommé *processor* dont le nombre correspond à celui de l'architecture physique ou logicielle. Ces étapes sont détaillées explicitement à la figure 2.2.

La description du placement se rapporte donc principalement à l'application ; car les directives du placement décrivent celui-ci indépendamment de l'architecture physique. L'alignement a pour but de donner le placement de tableaux relativement aux autres. Plusieurs stratégies d'alignement sont possibles. La figure 2.3 expose les uns aux autres .

Une fois l'alignement effectué, la distribution décrit la méthode de plongement du template sur le tableau de « *processors* ». Trois types de distributions sont aussi possibles : par bloc (`block`), circulairement (`cyclic`) et par bloc circulairement placé (`cyclic(`*b*`)`). En réalité, la dernière distribution généralise les deux premières. La distribution s'identifie par une fonction qui, à partir de l'adresse d'une cellule d'un *template*, détermine l'adresse d'une cellule de la structure *processor*. La

figure 2.3 décrit les 3 types de distribution sur un exemple.

2.2.2 Boucle parallèle

En OpenMP et en HPF, le parallélisme s'exprime par des structures de contrôle qui s'assimilent à des boucles. Cependant il s'agit plutôt d'un confort syntaxique car leur sémantique présente certaines différences à celle de la boucle séquentielle.

En OpenMP le parallélisme de la boucle se définit par une directive de compilation `#pragma parallel omp for` précédant la déclaration d'une boucle. Elle indique que l'ordre d'énumération des itérations est relaxé. Ainsi les tâches estampillées par les valeurs du compteur d'itérations peuvent s'exécuter en parallèle. Il s'agit d'un parallélisme de contrôle.

En HPF, la boucle `forall` correspond à une exécution différente. L'ordre d'exécution des itération est relaxée. De plus, chaque tâche travaille à partir d'une copie locale des données. Par conséquent, les données seront lues et copiées dans un espace appartenant à la tâche avant que le calcul opère. Il s'agit dans ce cas du parallélisme de données. D'autres particularités distinguent ces deux boucles dans le cas où elles s'appliquent non plus à une instruction mais à plusieurs instructions. La figure 2.4 résume les principales.

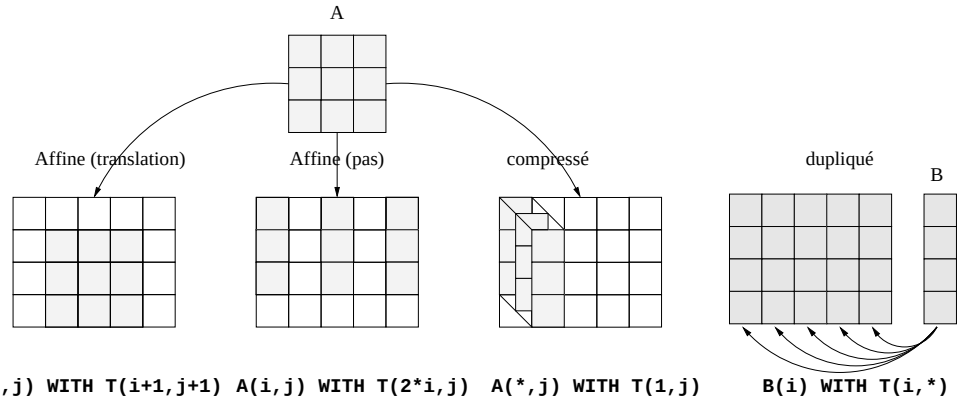
2.3 Principe de compilation des communications par vectorisation

La vectorisation des messages a pour but de regrouper le plus possible les données dans un seul message. Ce regroupement provient des données échangées provenant d'un tableau et pour une instruction données. Testée sur des programmes, cette optimisation procure l'accélération la plus importante parmi les optimisations possibles du code de communications [Tse93]. Nous définirons le schéma de compilation d'après une boucle parallèle correspondant à un paradigme de l'échange des données. Cette boucle décrit un nid de boucles imbriquées parallèles. Pour compiler ce programme, il faut communiquer les données nécessaires à une affectation. Les données en lecture seront transférées des processeurs les contenant aux processeurs conservant les données en écriture. Dans les programmes réels, il faut tenir compte du calcul. Une règle heuristique, nommée «le propriétaire calcule»[2], fixe le processeur réalisant le calcul comme étant celui où l'affectation s'effectuera. En respectant cette règle, la compilation des communications suivra donc le schéma décrit pour compiler celles du programme de la figure 2.5.

Les tableaux `A` et `B` sont de mêmes tailles et de mêmes dimensions. Pour simplifer, on prendra comme hypothèse que l'alignement fait correspondre les indices du tableaux à ceux du template par la fonction identité. La distribution sera une distribution bloc-cyclique sur deux représentations possibles de l'espace physique des processeurs. A partir de programmes plus complexes possédant un alignement non dupliqué, des tableaux de dimensions et de tailles différentes, on peut se rapporter à ce paradigme de communications. Dans la présentation du schéma de compilation nous confondrons la distribution du template et celle des tableaux. Précisons enfin que toutes les variables en minuscules $i, s, p, b, \cdots$ correspondent à des vecteurs. Les opérations s'étendent en effet naturellement aux vecteurs en appliquant celles-ci indépendamment sur chaque dimension.

A partir des directives de placement abordées dans la section précédente on caractérise deux fonctions $Owner_X$ et $Adloc_X$ qui déterminent respectivement l'adresse du processeur à partir de

[2] "Owner computes rule"



Alignement

la directive d'alignement en HPF décrit le placement les tableaux sur un espace *virtuel* de processeurs architecturés sous forme de tableau. Cet espace – nommé le *template* – a pour but de déterminer les positions des tableaux les uns relativement aux autres. La syntaxe de la directive d'alignement est : `ALIGN` $<$ *array spec* $>$ `WITH` $<$ *template spec* $>$
La spécification des alignements correspond à une fonction ou une relation qui, à partir de l'adresse d'une cellule d'un tableau, calcule celle(s) du template où la cellule du tableau sera placée. En HPF 3 types d'alignements différents sont autorisés : affine, compressé et dupliqué. Les deux premiers alignements correspondent à des fonctions affines. Le dernier correspond à une relation. Le schéma décrit les trois types d'alignement pour deux tableaux `A` et `B` sur un template `T`.

`T(BLOCK,BLOCK)` (`BLOCK` : $\&p = \&t \div c, c = |T| \div |P|$)

	0	1	2	3	4	5	6	7
0	$P_{(0,0)}$	$P_{(0,0)}$	$P_{(0,0)}$	$P_{(0,0)}$	$P_{(0,1)}$	$P_{(0,1)}$	$P_{(0,1)}$	$P_{(0,1)}$
1	$P_{(0,0)}$	$P_{(0,0)}$	$P_{(0,0)}$	$P_{(0,0)}$	$P_{(0,1)}$	$P_{(0,1)}$	$P_{(0,1)}$	$P_{(0,1)}$
2	$P_{(1,0)}$	$P_{(1,0)}$	$P_{(1,0)}$	$P_{(1,0)}$	$P_{(1,1)}$	$P_{(1,1)}$	$P_{(1,1)}$	$P_{(1,1)}$
3	$P_{(1,0)}$	$P_{(1,0)}$	$P_{(1,0)}$	$P_{(1,0)}$	$P_{(1,1)}$	$P_{(1,1)}$	$P_{(1,1)}$	$P_{(1,1)}$

`T(CYCLIC,CYCLIC)` (`CYCLIC` : $\&p = \&t \bmod |P|$)

	0	1	2	3	4	5	6	7
0	$P_{(0,0)}$	$P_{(1,0)}$	$P_{(0,0)}$	$P_{(1,0)}$	$P_{(0,0)}$	$P_{(1,0)}$	$P_{(0,0)}$	$P_{(1,0)}$
1	$P_{(0,1)}$	$P_{(1,1)}$	$P_{(0,1)}$	$P_{(1,1)}$	$P_{(0,1)}$	$P_{(1,1)}$	$P_{(0,1)}$	$P_{(1,1)}$
2	$P_{(0,0)}$	$P_{(1,0)}$	$P_{(0,0)}$	$P_{(1,0)}$	$P_{(0,0)}$	$P_{(1,0)}$	$P_{(0,0)}$	$P_{(1,0)}$
3	$P_{(0,1)}$	$P_{(1,1)}$	$P_{(0,1)}$	$P_{(1,1)}$	$P_{(0,1)}$	$P_{(1,1)}$	$P_{(0,1)}$	$P_{(1,1)}$

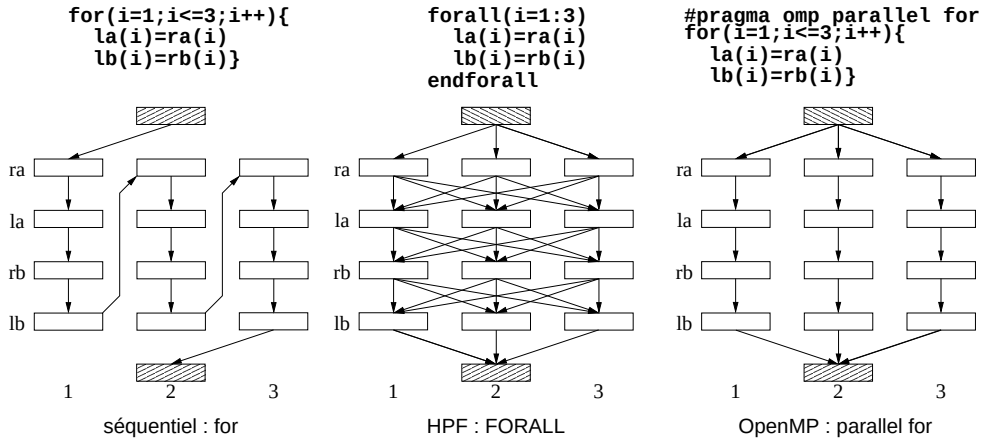
`T(CYCLIC(2),CYCLIC(3))` (`CYCLIC(`b`)` : $\&p = (\&t \div b) \bmod |P|$)

	0	1	2	3	4	5	6	7
0	$P_{(0,0)}$	$P_{(0,0)}$	$P_{(1,0)}$	$P_{(1,0)}$	$P_{(0,0)}$	$P_{(0,0)}$	$P_{(1,0)}$	$P_{(1,0)}$
1	$P_{(0,0)}$	$P_{(0,0)}$	$P_{(1,0)}$	$P_{(1,0)}$	$P_{(0,0)}$	$P_{(0,0)}$	$P_{(1,0)}$	$P_{(1,0)}$
2	$P_{(0,0)}$	$P_{(0,0)}$	$P_{(1,0)}$	$P_{(1,0)}$	$P_{(0,0)}$	$P_{(0,0)}$	$P_{(1,0)}$	$P_{(1,0)}$
3	$P_{(0,1)}$	$P_{(0,1)}$	$P_{(1,1)}$	$P_{(1,1)}$	$P_{(0,1)}$	$P_{(0,1)}$	$P_{(1,1)}$	$P_{(1,1)}$

Différentes distributions d'un TEMPLATE T(8,4) sur un PROCESSORS P(2,2)

La distribution a pour but de définir les règles de plongement d'un template sur une structure processors. Trois types de distribution sont possibles : par bloc (`block`) , circulaire ou cyclique (`cyclic`) et par bloc circulairement placé ou bloc-cyclique `cyclic(`b`)`. De manière sous-jacente, elles déterminent une fonction $Owner : \mathbb{Z}^n \mapsto \mathbb{Z}^n$ qui calcule l'adresse du processor correspondant en fonction de l'adresse d'une cellule d'un template. $|P|, |T|$ décrivent un vecteur contenant la dimension de la grille de processeurs. b est aussi un vecteur ($(2, 3)$ pour l'exemple). Les opérations de modulo (mod) et de division entière ($\div$) sont naturellement étendues sur les vecteurs en appliquant celles-ci indépendamment sur chacune des dimensions (alpha-extension). Dans cette figure, nous avons décrit le placement des cellules d'un template. La fonction *Owner* est décrite pour chacune des distributions. La distribution nommée `cyclic(`b`)` généralise les deux précédentes distributions. En effet la distribution par bloc correspond à $cyclic(|T|/|P|)$ et la distribution cyclique correspond à $cyclic(1)$. Sur cet exemple, on suppose que les tableaux débutent avec l'adresse 0.

FIG. 2.3 – Alignement et distributions en HPF



La figure décrit les dépendances à respecter entre les différentes opérations élémentaires à réaliser. Les rectangles à droite de `ra` et `rb` symbolisent les opérations de lecture et les rectangles à droite de `la` et de `lb` celles d'écriture. les rectangles hachurés situés en haut et en bas incarnent respectivement le point d'entrée et le point de sortie de la boucle. Les flèches entre deux opérations correspondent à des dépendances *définies par les règles d'exécution de la boucle* (et non pas par une analyse de dépendances). Rappelons qu'une opération s'exécutera uniquement si toutes les opérations la précédant ont été exécutées. Le schéma de gauche décrit les dépendances d'une boucle séquentielle. Le schéma du milieu, celui d'un `forall` : l'exécution d'une instruction pour toutes les itérations ne débute que si la précédente est achevée. Le schéma le plus à droite décrit celle d'une boucle parallèle OpenMP : le parallélisme est un parallélisme de contrôle. Au sein de chaque tâche exécutée en parallèle et estampillée à la valeur d'une itération (ie. $1, 2, 3$) l'exécution est séquentielle. Cette exécution correspond aussi à une variation du `forall` où l'on spécifie que l'exécution d'une itération est indépendante d'une autre (directive `$INDEPENDENT`). En résumé, le première boucle parallèle correspond à une séquence d'instructions parallèles (SEQ of PAR) alors que la seconde correspond à la composition parallèle de tâches séquentielles (PAR of SEQ) [Bou96].

FIG. 2.4 – Boucles & parallélisme

```
dimension(w) : : A,C
processors P(p), P'(p')
distribute C(cyclic(b)) onto P
distribute A(cyclic(b')) onto P'

forall (i ∈ D)
       A(f(i)) = C(g(i))
endforall
```

FIG. 2.5 – (prog.) Modèle de programme

celle d'une cellule d'un tableau X placée sur ce processeur et l'adresse locale de la cellule située dans la partie du tableau placé sur ce processeur. Soit b et p le vecteur de tailles d'un bloc (`cyclic(b)`) et le nombre de processeurs par dimension, on a :

$$Owner_A(x) = x \div b \bmod p \quad (2.1)$$
$$Adloc_A(x) = (x \div b^T.p) + x \bmod b \quad (2.2)$$

Pour qu'une communication identifiée par un couple de processeurs (s, s') soit valide , il faut vérifier que celle-ci décrive une affectation. Cette propriété de correction des communications définit l'ensemble des communications de la boucle ainsi :

Définition 2.1 (Ensemble de communications)

$$Comm = \{(s, s') \in [0 : p-1] \times [0 : p'-1] \mid \exists i \in \mathcal{D}.Owner(g(i)) = s \wedge Owner(f(i)) = s'\}$$

2.3.1 Schéma général

De cette définition, on constate que le rôle de chaque processeur pour chaque étape se détermine par rapport aux itérations de la boucle. Pour chaque processeur, il faut déterminer par rapport aux itérations les phases d'envoi, de calcul et de réception des données. Le programme généré sera un programme unique et générique qui est paramétré par l'adresse du processeur (s) (programme Single Program Multiple Data). En fonction de la valeur de ce paramètre, différentes exécution sont possibles à partir du même programme source.

Définition 2.2 (Itération locales) *Avec les notations du programme 2.5(On définit : $Ilocal_X(s)$ l'ensemble d'itérations de $\mathcal{D}$ conduisant à un référencement local de* `X` *sur le processeur d'adresse s. En fonction du programme 2.5, on a pour les variables A et C :*

$$Ilocal_A(s) = \{i \mid Owner_A(f(i)) = s \wedge i \in \mathcal{D}\}$$
$$Ilocal_C(s) = \{i \mid Owner_C(g(i)) = s \wedge i \in \mathcal{D}\}$$

Le calcul de ces ensembles permet de déterminer d'autres ensembles d'itérations et de communications nécessaires à la compilation des communications qui correspondent aux données à envoyer et aux données à recevoir. Si l'on considère un couple de processeurs (s, s'), les données à transférer se déduisent des itérations impliquant une communication entre ces deux processeurs. On a :

$$Icomm(s, s') = Ilocal_C(s) \bigcap Ilocal_A(s')$$

De cet ensemble, on peut déduire les deux aspects de la communication : celui vu de l'émetteur et celui vu du récepteur. Ces aspects correspondent respectivement à un calcul des adresses locales des données à lire pour constituer le message et des adresses locales pour affecter les données reçues. Les adresses des données à envoyer vers le processeur s' sont définies par :

$$Send(s, s') = \{c | c = g(i) \wedge i \in Icomm(s, s')\}$$

Et inversement les adresses des données devant être affectées localement sur s' des valeurs contenues dans le message sont données par :

$$Receive(s, s') = \{a | a = f(i) \wedge i \in Icomm(s, s')\}$$

L'application de la fonction *Adloc* (définition 2.2) aux adresses de ces précédents ensembles calcule les adresses correspondant à la mémoire locale du processeur concerné (s ou s'). D'une manière pratique, l'ordre d'énumération des itérations de *Icomm* doit être identique afin que le rangement des données dans le message corresponde à celui déterminé par la lecture. Le schéma de communication vectorisée est décrit par le programme 2.6. Ce programme correspond à un mode *Inspecteur-Exécuteur* où l'on prépare pendant la phase de l'inspecteur les calculs nécessaires à la communication qui sera réalisée pendant la phase exécuteur.

On notera enfin que $Icomm(s, s)$ décrit le calcul localement effectué. On définit l'ensemble des itérations conduisant à un calcul local ($ICompute(s)$) ainsi :

$$Icompute(s) = Icomm(s, s)$$

Ce schéma est général pour g, f et $\mathcal{D}$. Cependant, la définition de ce schéma – *centrée sur les itérations* – conduit à l'énumération complète de l'espace d'itérations sur chaque processeur pour constituer l'ensemble des itérations. Par conséquent, la complexité du calcul est identique à celle d'un calcul séquentiel. Toutefois, l'énumération porte uniquement sur un calcul entier des itérations et non pas sur les calculs en eux mêmes qui s'effectuent en parallèle. Ce schéma procure donc une accélération en parallélisant le calcul (représentée par l'affectation dans le programme 2.5). De plus le calcul des ensembles d'indices correspond à un coût additionnel de l'exécution en parallèle. Celui-ci n'est pas présent dans une version séquentielle. Pour accélérer ce traitement afin que ce calcul d'indices ne soit pas intégré il faut définir une optimisation. C'est l'objet de la section 2.3.2

2.3.2 Principe du schéma optimisé

Le principe d'un schéma optimisé des communications repose sur une énumération différente de celle des itérations. Cette énumération est *centrée sur les données*. L'objectif consiste à énumérer les adresses des données impliquées dans le calcul au lieu des itérations. De plus pour des raisons d'efficacité, cette énumération se déterminera uniquement par rapport au couple (s, s') d'adresses des processeurs impliqués dans la communication. Il n'y a donc pas à constituer d'ensembles en extension d'itérations ou d'adresses. Les techniques de passage entre un programme centré sur les itérations à un programme centré sur les données furent introduites par Corinne Ancourt [Anc91] dans un autre cadre applicatif.

Dans le cadre syntaxique des boucles à contrôle statique, nous avons proposé avec Cécile Germain [GD97] un schéma qui permet de modifier l'énumération des boucles de façon à ce que le calcul des ensembles *Ilocal* et *Icomm* soit inutile. Cette vectorisation est optimale (i.e. 1 message par couple de processeurs) pour une distribution par bloc ou pour une distribution cyclique. Pour les autres cas, l'énumération dépend d'une relation complexe entre la distribution et les fonctions référençant les données.

Schématiquement, cette vectorisation provient d'une triangularisation particulière d'un système d'équations et d'inéquations en nombre entiers. Ce système décrit les contraintes déduites de la propriétés de cohérence des communications et de la distribution des données. La triangularisation

```
Processeur s

                                        – Inspecteur –
for i ∈ D {
    Ilocal_A(Owner_A(f(i))) = Ilocal_A(Owner_A(f(i))) ∪ {i}
    Ilocal_C(Owner_C(g(i))) = Ilocal_C(Owner_C(g(i))) ∪ {i} }
for s' ∈ [0 : p − 1] { Icomm(s, s') = Ilocal_C(s) ⋂ Ilocal_A(s') }

                                        – Exécuteur –
for s' ∈ [0 : p − 1] − {s} {
    if (Icomm(s, s') ≠ ∅) {
       for i ∈ Icomm(s, s') { set(msg,C(Adloc_C(g(i))) }
       send(s,msg) } }

for i ∈ Icomm(s, s) { A(Adloc_A(f(i)))= C(Adloc_C(g(i))) }

for s' ∈ [0 : p − 1] − {s} {
    if (Icomm(s, s') ≠ ∅) {
       receive(s',msg) ;
       for i ∈ Icomm(s, s') { A(Adloc_A(f(i))) = get(msg) } } }
```

FIG. 2.6 – (prog.) schéma général de communications vectorisées

se traduit bien sûr par un nid de boucles. Il est réalisé de sorte à favoriser la constitution de messages vectorisés. Le cadre syntaxique statique est décrit dans le programme 2.7

La formulation du système incarne les contraintes déduites de ce programme. Pour formuler plus aisémment les contraintes, On définira :

- la matrice diagonale B : $B_{ii} = b_i, B_{ij} = 0$ si $i \neq j$ (idem B')
- la matrice diagonale P : $P_{ii} = p_i, P_{ij} = 0$ si $i \neq j$ (idem P')

Toute adresse d'un tableau peut se formuler par la composition d'une adresse d'un processeur s, d'une adresse m du bloc de taille b dans lequel cette donnée est rangée localement et enfin du déplacement d au sein de ce bloc. Cette formulation se déduit de la distribution. on a :

```
dimension(w) : : A,B
processors P(p), P'(p')
distribute C(cyclic(b)) onto P distribute A(cyclic(b')) onto P'

forall (Mi ≤ v)
       A(Ai + a) = C(Ci + c)
endforall
```

FIG. 2.7 – (prog.) Paradigme de programmes de communications

```
processeur s
for w ∈ W {
    s' = Φ(w, s) ;
    j = 0 ;
    for v ∈ V {
        mess[j++]=C(Ψ(w, v, s)) }
    if j > 0 { Send(mess,(s, s')) } }
```

FIG. 2.8 – (prog.) Schéma du programme d'envoi vectorisé

$$\begin{aligned} Owner_C(c) = s &\Leftrightarrow \exists m \in \mathbb{Z}^n, \exists d \in [0 : b-1], c = B(Pm + s) + d \\ Owner_A(a) = s' &\Leftrightarrow \exists m' \in \mathbb{Z}^n, \exists d' \in [0 : b'-1], a = B'(P'm' + s') + d' \end{aligned}$$

En regroupant l'ensemble des contraintes déduites de la propriété de cohérence des communications, et de celle de la distribution, on obtient le système d'équations et d'inéquations en nombre entiers suivants :

$$\begin{aligned} Ai + a &= B'(P'm' + s') + d' \\ Ci + c &= B(Pm + s) + d \\ Mi + v &\leq 0 \\ 0 &\leq d < b \\ 0 &\leq d' < b' \\ 0 &\leq s < p \\ 0 &\leq s' < p' \end{aligned}$$

Selon le programme d'envoi ou de réception, on aura une résolution différente. Celle-ci considérera comme paramètre s pour l'envoi ou s' pour la réception. L'autre paramètre sera calculé en fonction du premier. Classiquement, la triangularisation du système s'interprète ensuite comme un nid de boucles qui énumère les données à envoyer par processeur.

La difficulté de la résolution réside dans la nécessité de définir les boucles de manière à mettre en évidence la structure vectorisée. La résolution peut être trouvée dans [GD95, GD97]. Sans en détailler la technique, la structure finale du programme ressemblera à celle du programme 2.8. L'énumération des adresses est obtenue par la fonction Ψ. La fonction Φ détermine l'adresse du récepteur s'. Le schéma est optimal en nombre de messages si la fonction Φ est injective : à tout couple (w, s) correspond alors un seul s'.

Ce schéma a été validé par une maquette qui produit un code SPMD (PVM + C) de communications pour une grille 2D de processeurs à partir du programme paradigmatique présenté dans cette section. Il utilise la librairie OMEGA [Pug92] pour effectuer les calculs polyédriques.

2.4 Etat de l'art

Cet état de l'art reprend la division présentée dans [FC96]. Elle décrit les travaux en relation à la compilation des communications.

2.4.1 Les formes closes

De manière générale, la traduction de références déterminées dans un espace d'adressage unique en communication repose sur une formalisation ensembliste des itérations puis des adresses des données à transférer. Plusieurs formalisation sont possibles, nous en avons décrit une dans la section précédente. D'autres formalisations ont été proposées [FC96, Tse93]. Pour ces formalisations qui s'appliqueront à tout les programmes, la description des ensembles s'effectue en extension. Il faut donc conserver explicitement les valeurs des ensembles. Toutefois, dans des cas particuliers mais fréquemment rencontrés dans les programmes scientifiques, ces ensembles possèdent une régularité qui permet de définir des représentation concises – *en intention* – pour les représenter. De plus, les opérations nécessaires au calcul sont closes, c'est-à-dire qu'elles permettent de décrire l'ensemble résultat avec le même formalisme que celui des ensembles en paramètres de l'opération. Bien entendu, cette formalisation s'applique dans un cadre syntaxique particulier. L'une des modélisations employées fréquemment est la description par *section de tableaux*. On définit une section de tableau $[l : u : s]$ ainsi :

$$[l : u : s] = \{i | i = l + sj, j \in \mathbb{N}, (s > 0 \wedge i \leq u \vee s < 0 \wedge i \geq u\}$$

Les sections de tableaux sont closes par intersection ($\cap$) et par transformation affine. Elles correspondent de plus à une construction syntaxique rencontrée en HPF et FORTRAN90 [MR90] car cette formulation permet de décrire des calculs parallèles *« simples »*. Par exemple un décalage à droite s'exprime ainsi : `A[2:11]=A[1:10]`. Leur intérêt réside donc à la fois dans leur capacité à décrire des espaces de données référencés et dans la simplicité des calculs sous-jacent à la modélisation. L'utilisation des sections de tableaux permet de valider simplement et de manière crédible les techniques de compilation de communications [Koe91] car ils correspondent à des références fréquemment rencontrées dans les programmes. Toutefois, elles couvrent un champ syntaxique plus restreint que les polyèdres convexes (sous section 2.4.3).

2.4.2 Automate à état fini

L'accès à des sections de tableaux peut conduire à une formulation complexe qui nécessite d'autres outils pour capturer cette complexité efficacement. Chatterjee [CGST93] traite le cas des accès à des sections de tableaux distribués par bloc-cyclique. Schématiquement, cette distribution complexifie le parcours des espaces locaux d'indices car il faut traduire un parcours dans la mémoire locale en un parcours aux sein d'un bloc et un parcours de blocs en blocs. Le compilateur génère ce parcours en créant un automate à états finis où chaque état est valué par le déplacement à effectuer pour référencer localement la prochaine donnée impliquée dans le calcul. Cet automate est aussi paramétré par l'adresse du processeur sur lequel il s'exécute.

2.4.3 Polyèdres et treillis

L'approche la plus générale qui étend naturellement la description par section de tableaux est celle utilisant les polyèdres convexes [Shr86] ou les formules de Presburger. Les propriétés et condi-

tions se décrivent par des systèmes d'équations et d'inéquations en nombre entiers. Dans le cadre syntaxique des programmes à contrôle statique, c'est certainement celle qui offre un pouvoir de formalisation et de généralisation le plus intéressant. De plus, la résolution des systèmes ou leur triangularisation s'interprètent sous forme de programmes à nids de boucles, ce qui permet une génération de code optimisée sophistiquée. Le parcours des espaces locaux en utilisant uniquement des informations locales aux processeurs constituent un des points critiques des méthodes de compilation de communications. Pour des transformations non singulières, Ramajunan [Ram92] présente une méthode de parcours fondées sur la décomposition d'une matrice sous forme normale de Hermite qui permet d'effectuer des parcours non triviaux des espaces locaux.

F. Coehlo [Coe96] généralise cette approche dans le cadre des programmes parallèles à contrôle statique en intégrant tous les types de distributions qui sont spécifiés dans HPF.

J.L. Pazat et M. Lefur [FPA95, Paz97] utilisent une technique d'énumération fondées sur les outils du calcul polyédrique mais en parcourant les espaces d'itérations et non pas les espaces d'adresses des tableaux. Le *« découpage »* opéré de l'espace d'itération se détermine à partir de la spécification du placement en C-PANDORE. Avant d'effectuer un placement final d'un tableau sur les processeurs, celui-ci est préalablement découpé en bloc. Puis ces blocs sont placés physiquement sur l'architecture selon une stratégie par bloc (de blocs) ou cyclique. Ce placement peut se rapporter au placement bloc-cyclique. La génération du code de communications est générique et indépendant du placement final des blocs sur les processeurs.

2.5 Bilan

La caractérisation d'une problématique unifiée pour le routage statique et pour l'optimisation des communications par vectorisation de messages ont permis d'étendre et de généraliser les travaux entrepris dans le cadre de ma thèse à la compilation des langages à parallélisme de données pour des grappes de PCs. L'analyse a été développée pour la distribution cyclique, puis étendue ensuite pour le cas général `cyclic(`b`)`.

Cette étude a fait l'objet de la réalisation de 2 maquettes. La première concernait un compilateur pour une langage HPT, sous-ensemble de HPF. Cette maquette produisait un code exécutable pour un simulateur fonctionnel de l'architecture PTAH à 64 processeurs. Il fut bâti en modifiant le compilateur TINY de M. Wolfe. Ce compilateur comportait 15000 lignes. Ce développement s'est effectué en collaboration avec 2 étudiants de 3^{me} cycle, M. Pic et R. Carlier

L'autre réalisation, faite à la suite de ma thèse concernait un compilateur de communications, qui permettait de valider la génération automatique de code de communications compilées. Il produisait automatiquement un programme de communications en utilisant pour les calculs symboliques la librairie OMEGA [Pug92]. Ces deux recherches ont été menées en collaboration avec Cécile Germain et ont fait l'objet d'un des thèmes de l'action PARADIGME et du GDR ANM .

Chapitre 3

Traitements de parallélisation pour les structures creuses

3.1 Introduction

De nombreuses méthodes d'optimisations comme la compilation des communcations se fondent sur l'utilisation des polyèdres convexes [Fea91, FR88, Wol96]. Leur intérêt n'est à présent plus à démontrer. Cependant, cette utilisation n'est possible que pour un sous ensemble de programmes pour lesquels la modélisation peut s'appliquer. Ces programmes correspondent à un sous ensemble des constructions possibles du langage. Ils se nomment les *programmes à contrôle statique* [Fea91] appelés aussi *programmes réguliers*. Il apparaît donc intéressant de vouloir dépasser les restrictions imposées afin d'optimiser des programmes qui ne sont pas à contrôle statique.

L'extension proposée a pour objectif d'analyser des programmes contenant des fonctions d'accès aux tableaux et des fonctions de bornes de boucles qui ne sont plus affines, comme dans le cas régulier. Cette irrégularité couvre un large spectre d'algorithmes. Nous nous sommes donc restreints à une certaine forme d'irrégularité, celle des algorithmes utilisant une structure particulière, les structures creuses.

Les structures creuses correspondent à des structures de données représentant des tableaux compressés où la valeur 0 est éliminée du stockage. A cause de ce format, les algorithmes sortent du cadre régulier car ils intègrent dans l'expression des accès, la compression inhérente à cette élimination. Celle-ci se traduit par l'utilisation d'adressage indirect. A titre d'exemple, le programme 3.1 décrit un algorithme de multiplication de matrice-vecteur avec une matrice *« dense »* et avec une matrice creuse. On constate que la violation du cadre réguler s'applique à la fois sur les bornes des boucles qui ne sont plus des fonctions affines ayant pour variables les index des boucles, et sur les

Propriétés de l'élément e		Exemple pour 0
neutralité à gauche	$\forall a \in X, a = e \oplus a$	`+`
neutralité à droite	$\forall a \in X, a = a \oplus e$	`+, -`
absorption	$\forall a \in X, e = a \otimes e = e \otimes a$	`*`
point fixe	$e = \mu(e)$	$\sqrt{x}$

La deuxième colonne de cette table décrit les propriétés algébriques énoncées dans la première colonne de l'élément e. La dernière colonne donne des exemples d'opérations où l'élément 0 respecte ces propriétés.

TAB. 3.1 – Propriétés

références aux tableaux qui sont d'autres tableaux ; entraînant un calcul d'adresse par indirection a priori non-prévisible.

L'irrégularité dépend uniquement du choix de la structure de données employée qui représente une forme particulière de stockage d'un vecteur ou d'une matrice. Une structure creuse peut être alors considérée comme une structure de données adaptée à la spécificité des données. En effet, les algorithmes dédiés aux structures creuses possèdent leur équivalent pour des tableaux.

Ainsi, définir un programme pour une structure de données creuse correspond à transformer un programme destiné à des tableaux multi-dimensionnels en un programme destiné à une structure creuse. Cette conversion peut s'assimiler dans sa démarche à la linéarisation des accès pour implanter des tableaux à plusieurs dimensions dans un tableau à une dimension représentant la mémoire centrale ; conversion qui est pratiquée par tous les compilateurs. Cette assimilation montre qu'il s'agit d'une pratique classique de conversion d'une structure de données décrite par l'utilisateur dans un format approprié à l'implantation finale. Malgré sa complexité supérieure à la linéarisation, ce procédé peut s'automatiser. Les travaux décrits dans l'état de l'art (section 3.4) montrent comment cette approche peut être menée pour un programme séquentiel. Transformer un programme portant sur les tableaux en un programme portant sur les structures creuses se nomme l' *éparpillement* [1].

Ce procédé a pour effet de permettre de nouveau d'optimiser ces programmes car ils sont décrits pour des tableaux denses dans une formulation correspondant aux programmes à contrôle statique. Charge au compilateur d'en effectuer l'éparpillement. Ce procédé permet aussi de simplifier la programmation en déléguant au compilateur la tâche d'implantation des tableaux en structures creuses, tâche souvent considérée comme délicate.

Cette approche s'inscrit dans la même logique que celle développée pour la compilation des communications où l'utilisateur avait l'illusion de définir des accès à une mémoire unique.

Toutefois, si l'outil d'analyse de programme est conservé, la problématique a changé : il s'agit d'opérer des optimisations ayant pour cible des programmes dédiés aux structures creuses. Nous avons proposé une nouvelle méthode de parallélisation appelée *« parallélisation creuse »*. Avant de présenter la démarche sous-jacente à cette méthode nous présentons l'état de l'art dans ce domaine (section 3.4) puis nous décrirons le principe général (section 3.1) .

La parallélisation creuse trouve son origine dans les propriétés particulières du zéro (élément neutre pour l'addition et élément absorbant pour la multiplication) dont l'utilisation a déjà prouvé son efficacité pour l'algorithme de factorisation de Cholesky creux [HNP91], [GKK97]. C'est pour-

[1] traduction libre du néologisme anglais «sparsification»

```
1   do i=1,n
      do j=1,m
        V(i)=V(i)+A(i,j)*U(j)
      enddo
5   enddo
```

```
1   do i=1,n
      do j=RO(i),RO(i+1)-1
        V(i)=V(i)+DA(j)*U(CO(j))
      enddo
5   enddo
```

$$A = \begin{pmatrix} 0 & \mathbf{-8} & 0 & \mathbf{2} & 0 \\ 0 & \mathbf{8} & \mathbf{3} & 0 & 0 \\ 0 & 0 & \mathbf{-4} & \mathbf{4} & 0 \\ \mathbf{9} & 0 & 0 & 0 & \mathbf{1} \\ 0 & \mathbf{11} & 0 & 0 & 0 \end{pmatrix}$$

Format dense

DA
-8
2
8
3
-4
4
9
1
11

CO
2
4
2
3
3
4
1
5
2

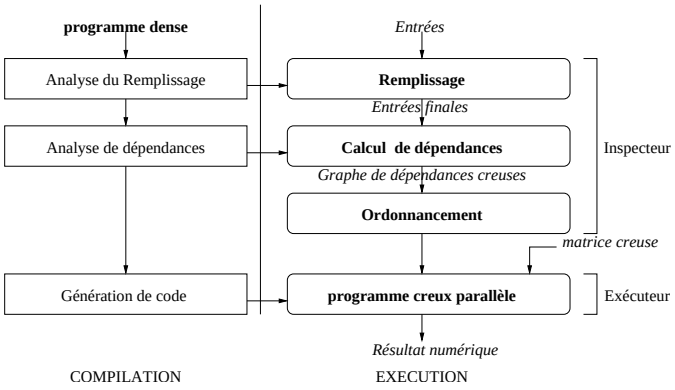
RO
1
3
5
7
9
10

Format creux CSR

La représentation CSR d'une matrice correspond à la compression en suivant les lignes de la matrice (Compress Sparse Row). Cette compression regroupe toutes les données non nulles de la matrice `A` (`DA`). Pour restaurer l'espace dense afin de permettre le calcul avec des tableaux non compressés comme c'est le cas pour la multiplication d'une matrice creuse avec un vecteur dense, la structure est dotée d'informations `CO` et `RO` permettant cette restauration. La structrue de données est conçue pour favoriser un parcours particulier. Dans ce cas, la boucle la plus interne parcourt les lignes de la matrice. Les rôles des tableaux de cette structure creuse sont :

- `DA` : Cette table conserve les valeurs non nulles de la matrice. L'ordre de stockage correspond à un parcours lexicographique des couples (ligne,colonne).
- `CO` : Cette table conserve les indices des colonnes de chaque valeur conservée dans `DA`. Une valeur `DA(`j`)` possédait initialement la position `CO(`j`)` dans la ligne. Cette table joue un rôle clé pour des calculs mélangeant structure dense et structure creuse.
- `RO` : Cette table décrit en réalité des intervalles qui déterminent quelles sont les valeurs dans `DA` qui correspondent à une ligne données. Les valeurs non-nulles de la ligne i de la matrice `A` correspondent à celles conservées dans `DA(`j`)`, avec `RO(`i`)`$\leq j \leq$`RO(`$i+1$`)`-1. Cette table possède un nombre de cellule correspondant au nombre de lignes augmenté de 1 ($|RO| = |A_{(i,*)}| + 1$) ; car les algorithmes de parcours par ligne le nécessitent ainsi (cf. matrice vecteur creux).

FIG. 3.1 – Multiplication de matrice-vecteur dense et creuse (format CSR)



Ce schéma résume les étapes de compilation. Chaque étape génère un code spécifique pour l'inspecteur qui sera exécuté en préambule à l'éxécuteur afin de calculer l'ordonnancement des tâches de l'exécuteur. Ces étapes sont :

- Le remplissage dans l'inspecteur détermine symboliquement les *entrées* correspondant aux coordonnées des valeurs devenant différentes de e durant l'exécution. Ces entrées sont indispensables aux calculs creux car, non seulement elles permettent de calculer les dépendances, mais elles offrent aussi la possibilité de réserver au plus juste la mémoire nécessaire aux valeurs numériques. La fonction de remplissage sera générée à la compilation.
- Le calcul de dépendances construit un graphe de dépendances que nous appelons le *graphe de dépendances creuses*. Sa construction est définie par des contraintes affinant les conditions de dépendances de Bernstein [Ber66].
- La phase d'ordonnancement séquence dynamiquement les tâches du programme en respectant l'ordre donné par le graphe de dépendances creux. Les tâches qui seront exécutées correspondent à un programme traduit en un programme creux.
- La génération de code creux transcrit un programme écrit pour des tableaux en un programme adapté aux structures creuses.

FIG. 3.2 – Étapes de la compilation et de l'exécution

quoi, nous visions à la généraliser et automatiser son traitement. Ceci conduit à étendre les tests de dépendances classiquement utilisés en parallélisation automatique afin d'identifier formellement les causes de ce parallélisme. Bien qu'initialement inspiré du comportement particulier du zéro vis-à-vis des opérations flottantes, nous considérerons plus généralement un élément e qui vérifie les propriétés énoncées et détaillées dans la figure 3.1.

La méthode de parallélisation que nous présentons est *semi-statique* dans le sens où elle requiert premièrement une analyse à la compilation faite sur le texte du programme et deuxièmement une analyse à l'exécution portant sur la structure des matrices creuses. Le mode d'exécution est de type inspecteur-exécuteur : l'inspecteur préparera l'exécution en calculant en parallèle le graphe de dépendances afin de déterminer ensuite un ordonnancement. l'exécuteur effectuera le calcul numérique creux en parallèle selon cet ordonnancement. Un inspecteur correspond généralement à un module de traitement dynamique identique pour tous les programmes. Dans notre cas, l'inspecteur est spécialisé au programme analysé et est généré par le compilateur. Pour calculer dynamiquement l'ordonnancement, l'inspecteur exécutera auparavant les fonctions spécialisées de calcul du remplissage et de calcul des dépendances générées à la compilation. La figure 3.2 décrit plus précisement

```
1    do I=2,6
2
 A(I+1)= (A(I-1)*A(I-2))-A(I+1)
3    enddo
```

FIG. 3.3 – (prog.) Exemple

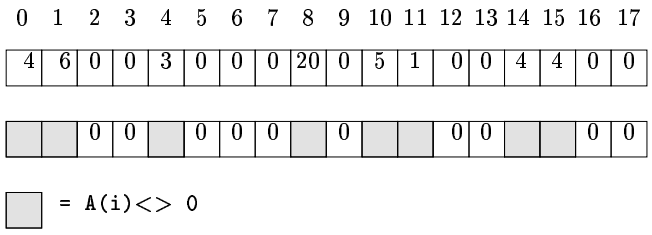


FIG. 3.4 – Remplissage initial

les étapes de ce processus.

Considérons le programme 3.3 appliqué au tableau de la figure 3.4. Notre but est d'illustrer l'analyse de dépendances selon deux modes : celui fait par une analyse de dépendances selon les conditions de Bernstein et celui fait en considérant le programme *et* la structure de la matrice que nous appellerons *l'analyse creuse*.

Une instruction s dans une boucle donnera lieu à l'exécution de plusieurs opérations. Une opération que l'on notera $s(i)$ est repérée par une instruction s étiquetée par les valeurs i des indices des boucles. Une dépendance de flot traduit une relation producteur-consommateur entre deux opérations. Deux opérations $s(i)$ et $s'(i')$ du programme 3.3 sont dépendantes par flot si $s(i)$ s'exécute avant $s'(i')$ et le résultat écrit lors de l'opération $s(i)$ est lu lors de l'exécution de l'opération $s'(i')$. Dans notre exemple ne comportant qu'une seule instruction ($s = s'$), ces opérations sont dépendantes si la cellule mémoire $A(i+1)$ accédée en écriture par l'expression `A(I+1)` lors de l'opération $s(i)$ doit être lue par l'opération $s(i')$. Nous supposerons, par exemple, que cette cellule correspond à $A(i'-1)$.

La condition de dépendances entre ces opérations est donc : $i+1 = i'-1 \wedge i < i'$. L'ensemble des couples (i, i') d'itérations symbolisant les dépendances entre les opérations du programme 3.3 vérifiant cette condition est :

$$\{(2,4),(3,5),(4,6),(5,7),(6,8),(7,9),(8,10),$$
$$(9,11),(10,12),(11,13),(12,14),(13,15),(14,16)\}$$

(on a omis d'indiquer l'instruction s pour repérer les opérations puisque dans notre exemple, il y a une seule instruction dans le corps de boucle).

Les conditions définies par Bernstein [Ber66], qui permettent de calculer les dépendances précédentes, prennent uniquement en compte l'accès commun à une cellule mémoire et non pas la valeur d'une cellule ni les opérations appliquées sur ces cellules. Nous allons voir comment la propriété d'absorption $0 * n = 0$ fait que certaines dépendances peuvent être éliminées. Prenons le calcul de $s(7)$; en constatant que $A(5) = 0$ durant toute l'exécution du programme, son calcul devient :

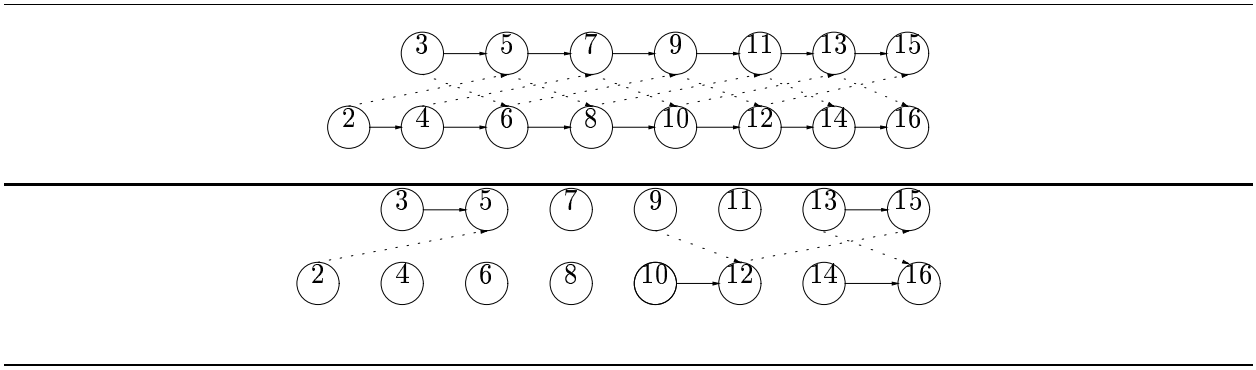


FIG. 3.5 – Graphe de dépendances du programme de la fig. 3 selon les deux analyses

$$A(8) = (A(6) * A(5)) - A(8)) = A(6) * 0 - A(8) = -A(8)$$

On peut en déduire que $s(7)$ ne dépend pas pour le calcul de $A(8)$ de l'opération $s(5)$ car $A(6)$ n'intervient pas pour le calcul de $A(8)$ d'après la propriété d'absorption ; ce qui conduit à la suppression de la dépendance $(5, 7)$. De même, dans la mesure où la valeur $A(5)$ demeure identiquement nulle, nous considérerons que $s(4)$ ne produit pas de valeur. La dépendance $(4, 7)$ sera aussi supprimée.

En raisonnant de la même manière, on obtient l'ensemble des couples de dépendances entre itérations d'un tableau creux :

$$\{(3, 5), (10, 12), (13, 15), (14, 16)\}$$

Cet exemple met en évidence l'intérêt d'affiner l'analyse de dépendances en tenant compte de la structure creuse du tableau. Mais pour cela, il a fallu déterminer que $A(5)$ reste identiquement nul au cours du calcul.

Le premier graphe de la figure 3.5 représente le graphe de dépendances obtenu d'après les conditions de Bernstein et le second le graphe de dépendances est celui déduit de ce calcul. Dans ces graphes, les traits pleins représentent les dépendances induites par la référence `A(I+1)` en écriture et `A(I-1)` en lecture et ceux en pointillés, la dépendance induite par les références `A(I+1)` et `A(I-2)`.

En appliquant un ordonnancement au plus tôt par un algorithme de tri topologique des deux graphes de dépendances, on obtient les ordonnancements décrits par la table 3.2. Dans le second ordonnancement, on a aussi supprimé les tâches $\{4, 6, 8, 11\}$ ne conduisant à aucun calcul. L'itération 7 demeure car le calcul $A(8) = -A(8)$ modifie le contenu de la cellule. L'analyse creuse réduit le temps de l'algorithme de 5 étapes.

3.2 Principe de la parallélisation creuse

Dans cette section nous résumerons uniquement le principe en s'appuyant sur des exemples simples portant sur des calculs scalaires pour des blocs d'instructions. L'extension aux tableaux est complètement décrite dans les articles suivants : [AD99, AAD00].

Afin d'introduire le raffinement proposé pour les dépendances creuses, nous rappellerons brièvement le principe du calcul des dépendances. Pour définir les dépendances, nous adopterons la définition donnée par P. Feautrier [Fea92a] de l'exécution en parallèle de deux opérations. Deux

	Ordonnancement	
date	dépendances denses	dépendances creuses
1	2, 3	2, 3, 7, 9, 10, 13, 14
2	4, 5	5, 12, 16
3	6, 7	15
4	8, 9	
5	10, 11	
6	12, 13	
7	14, 15	
8	16	

TAB. 3.2 – Ordonnancement du programme de la figure 3

opérations n'accédant pas en écriture aux mêmes variables s_1 : $\mathtt{x_1} = \mathtt{expr_1}$ et s_2 : $\mathtt{x_2} = \mathtt{expr_2}$ peuvent être exécutées en parallèle si elles commutent. C'est-à-dire que l'exécution $s_1; s_2$ donne le même résultat pour $\mathtt{x_1}, \mathtt{x_2}$ que l'exécution $s_2; s_1$.

Une condition suffisante de parallélisation choisie par Bernstein est obtenue en regardant si les instructions n'accèdent pas aux mêmes variables du programme. On définit le prédicat $Bernstein(s_1, s_2)$ par :

$$Bernstein(s_1, s_2) = (R(s_1) \cap W(s_2) = \emptyset) \wedge (R(s_2) \cap W(s_1) = \emptyset) \wedge (W(s_1) \cap W(s_2) = \emptyset)$$

où R, W correspondent respectivement aux ensembles des variables en lecture et en écriture.

Soit le prédicat $Par(s_1, s_2)$, qui est vrai si l'exécution de $s_1; s_2$ commute, on a l'implication suivante $Bernstein(s_1, s_2) \Rightarrow Par(s_1, s_2)$ ou de manière équivalente :

$$\neg Par(s_1, s_2) \Rightarrow \neg Bernstein(s_1, s_2) \tag{3.1}$$

La relation de dépendance entre opérations sera caractérisée par le prédicat Dep indiquant une violation des conditions de Bernstein, soit :

$$Dep(s_1, s_2) = (s_1 \prec s_2) \wedge \neg Bernstein(s_1, s_2) \tag{3.2}$$

De l'implication 3.1, on en déduit que toutes les opérations vérifiant $s_1 \prec s_2$ qui ne commutent pas vérifient nécessairement $Dep(s_1, s_2)$. Mais, des opérations qui commutent peuvent aussi vérifier Dep[2]. Il faut enfin préciser que d'autres facteurs, dus notamment aux outils mathématiques d'analyse, peuvent aussi être des causes d'approximation. Les ouvrages [Ban88] et [Wol96] offrent une présentation plus approfondie sur l'analyse de dépendances.

En fonction de l'intersection des ensembles en lecture et en écriture invalidant le prédicat $Bernstein$, on qualifiera différents types de dépendances Par exemple, pour le programme suivant (3.6) on a une dépendance de flot[3] car $W(s_1) = \{\mathtt{Y}\}$ et $R(s_2) = \{\mathtt{Z}, \mathtt{X}, \mathtt{Y}\}$:

[2]Par exemple, `a=5 ; b=a*0+10` donne le même résultat que `b=a*0+10 ; a=5` bien que `a` soit commun aux deux instructions.

[3]Rappelons que cette dépendance se définit ainsi : $s_1 \delta s_2 = s_1 \prec s_2 \wedge W(s_1) \cap R(s_2) \neq \emptyset$

```
0   X=0;
1   Y=1;
2   Z=Z+1+X*Y;
```

FIG. 3.6 – (prog.) Exemple scalaire

```
1   X=0;
2   Y=1;
3   Z=Z+1+0*0;
```

FIG. 3.7 – (prog.) Exemple scalaire après substitution

3.2.1 Creuser les dépendances

L'analyse creuse raffine la définition des dépendances afin de réduire le nombre de dépendances en excès. Intuitivement, on s'aperçoit de la possibilité d'utiliser l'élément e pour supprimer des dépendances. La difficulté réside dans la formalisation de cette propriété. La démarche adoptée est de caractériser la possibilité de substituer à une variable en lecture la constante e dans une expression tout en conservant le même résultat pour le calcul. On définira la substitution par :

Définition 3.1 *Etant données une constante e et une expression* `exp`*, soit* $[\![exp]\!]\rho$ *l'évaluation de cette expression dans un environnement* ρ *définissant l'état des variables du programme, on caractérisera la propriété de substitution d'une sous expression* `exp'` *de* `exp` *par e dans un environnement* ρ *ainsi :*

$$subst(\texttt{exp}, \texttt{exp}', \rho) = ([\![exp]\!]\rho = [\![exp\ \ [e/\texttt{exp'}]]\!]\rho)$$

où $exp\ [e/\texttt{exp'}]$ *désigne l'expression* `exp` *dans laquelle les occurrences de* `exp'` *ont été remplacées par e.*

Par exemple pour le programme précédent, on peut substituer dans l'expression de s_2 `X` par 0 car sa valeur est nulle, mais on peut aussi substituer `Y` par 0 car $\forall y, 0.y = 0.0 = 0$. le programme 3.7 représente ces différentes substitutions. Le résultat est identique au programme 3.6 mais il est aisé de constater que l'on peut exécuter en parallèle les instructions.

De manière pratique, la substitution sera valide dans deux cas : le premier cas est trivial, il s'agit de celui où l'état d'une variable est e lors de l'évaluation de l'expression. C'est le cas de `X` dans le programme 3.6. Le second cas repose sur la non injectivité de certaines fonctions (expressions) ; c'est-à-dire que l'on a $h(x, y) = h(x, e)$ et $y \neq e$. Ce cas peut survenir en fonction d'une combinaison d'autres valeurs de variables. C'est le cas de `Y` dans le programme 3.6 car 0 est absorbant pour la multiplication.

Afin de profiter du parallélisme introduit par la substitution, on affinera la définition d'une dépendance de flot, $s_1 \delta s_2$, en y ajoutant une condition supplémentaire imposant l'impossibilité d'effectuer une substitution. Cette définition modifie donc les conditions de dépendances.

$$\begin{aligned} s_1 \delta^S s_2 \equiv \quad & (s_1 \prec s_2) \wedge (W(s_1) \cap R(s_2) \neq \emptyset) \wedge \\ & (\exists \mathtt{a} \in W(s_1) \cap R(s_2), \neg subst(\mathtt{exp_2}, \mathtt{a}, \rho^{s_2})) \end{aligned}$$

où ρ^{s_2} désigne l'environnement correspondant à l'état des variables avant l'exécution de s_2. Une amélioration immédiate concerne l'élimination des dépendances pour les variables qui possèdent

une valeur égale à e. Si l'on connaît a priori l'ensemble E^{s_2} des variables qui posséderont une valeur différente de e avant l'exécution de s_2, il est possible de restreindre l'ensemble des variables à examiner uniquement à celles-ci. La nouvelle définition des dépendances sera :

$$\begin{aligned} s_1 \delta^S s_2 \equiv \quad & (s_1 \prec s_2) \wedge (W(s_1) \cap R(s_2) \cap E^{s_2} \neq \emptyset) \wedge \\ & (\exists \mathtt{a} \in W(s_1) \cap R(s_2) \cap E^{s_2}, \neg subst(\mathtt{exp_2}, \mathtt{a}, \rho^{s_2})) \end{aligned}$$

Pour s_2 (programme 3.6), comme $E^{s_2} = \{\mathtt{Z}, \mathtt{Y}\}$, le test vérifiera la propriété de substitution pour `Y`.

3.2.2 Résumé de l'extension aux tableaux

L'extension aux tableaux et aux boucles de ce principe a pour objet de construire le graphe de dépendances creuses des opérations de cette boucle.

L'une des principales difficultés consiste alors à transcrire l'interdiction de substituer une variable par la constante e car cette propriété implique de connaître chaque adresse de cellules différentes de e. Son extension repose donc sur la possibilité de définir formellement et automatiquement les cellules qui seront remplies au cours de l'exécution puis à partir de ce remplissage d'en déterminer l'extension du calcul sur la substitution.

Pour cela, on applique une méthode reposant sur deux notions : une interprétation abstraite du calcul et une capture des adresses des cellules non nulles qui découlent de l'interprétation.

Les instructions seront interprétées en considérant un domaine de valeurs booléénnes. La valeur vraie symbolise le fait qu'une cellule sera différente de e alors que la valeur fausse désigne une cellule qui sera égale à e. De ce fait, la sémantique de ce calcul fournit un résultat booléen traduisant la création d'une valeur.

En considérant l'exemple choisi et en posant $e = 0$, le calcul de l'instruction `A(i+1)=A(i-1)* A(i-2)-A(i+1)` est équivalent à `A(i+1)=A(i-1)` $\wedge$ `A(i-2)` $\vee$ `A(i+1)` car il est nécessaire de `A(i-2)` et `A(i-1)` soient différents de 0 pour que le résultat soit différent de 0 . En se fondant sur ce même raisonnement, on identifiera l'addition et la soustraction au ou logique. On s'aperçoit qu'il s'agit d'une approximation car on ne tient pas compte du fait que deux valeurs non nulles peuvent produire une valeur nulle.

Cette interprétation abstraite relativement naturelle constitue une explication simplifiée du calcul effectif car le remplissage opère en réalité uniquement sur des adresses correspondant à des valeurs non nulles. En effet la représentation des adresses correspond à un format creux qui se symbolise par un ensemble d'entiers alors qu'un tableau de booléens correspond à une structure dense. Il s'agit donc de calculer l'adresse cellules qui deviendront non nulle à partir de l'adresse des cellules que l'on a déjà identifiées comme étant non nulles. Cet ensemble incarnent les entrées.

Sur l'exemple cité, on considérera trois adresses a_1, a_2, a_3 telles que pour une itération i donnée, on ait : $a_1 = i + 1, a_2 = i - 1, a_3 = i - 2, 2 \leq i \leq 16$. Il faut alors définir une relation entre a_1, a_2, a_3 définissant les cellules à remplir. Par substitution de variables, on en déduit que sous la condition $a_3 = a_2 - 1$ la cellule produite sera $a_1 = a_2 + 2, 1 \leq a_2 \leq 15$. Cette opération se détermine automatiquement en s'appuyant sur des outils du calcul polyhédrique. Les articles [AD99, AAD00] détaillent ce calcul plus en profondeur. Le calcul de toutes les adresses qui seront remplies lors du calcul effectif s'obtient en itérant le processus consistant à appliquer les formules obtenues sur l'ensemble des adresses déjà présentes. Ce processus est itéré jusqu'à sa convergence. Nous avons prouvé que le calcul du remplissage définit une fonction monotone et continue. Dans ce cadre,

l'obtention d'un plus petit point fixe est toujours vérifié. De même, la propriété de non substitution doit être définie par rapport aux entrées. De nouveau celle-ci sera caractérisée par une interprétation abstraite déterminant, par un prédicat, les conditions de non substitution.

La définition des dépendances creuses par le biais de la propriété de non substitution représente la formalisation de la parallélisation creuse par des méthodes ad-hoc. En effet, la définition des dépendances obtenues pour l'algorithme de factorisation de Cholesky correspond à celle trouvée de manière ad-hoc (cf. [AD99, AAD00, Adl99]).

3.3 Génération de code

Le verrou spécifique à la génération de code pour la parallélisation creuse réside dans l'intégration de l'éparpillement à la parallélisation. Cette section résume la manière dont cette intégration se réalise. De manière générale, la parallélisation définit un nouvel ordonnancement des itérations en fonction du graphe de dépendances obtenus. La structure du programme généré à partir d'un programme séquentiel est décrit par le programme 3.8. L'ordonnancement décrit ici est un ordonnancement statique par fronts pour un modèle PRAM. Précisons cependant que d'autres stratégies peuvent s'appliquer [FY97] telles qu'un ordonnancement dynamique [SMC91] (self-scheduling). Le programme $\texttt{P}(\vec{I})$ possède les caractéristiques d'un programme séquentiel ; ce qui respecte le cadre d'analyse de l'éparpillement. Plus précisément, le flot de contrôle du programme `P` est séquentiel. L'utilisation d'un langage possédant un espace d'adressage unique correspond à l'organisation mémoire dans un langage séquentiel. L'exécution des différentes instances de `P`, réglée par l'ordonnanceur, garantit la correction de l'exécution par rapport aux contraintes de précédence décrites par le graphe de dépendances. En résumé, avec les hypothèses prises par les études d'éparpillement, un module de conversion du dense vers le creux peut s'intégrer en complément du traitement de parallélisation creuse de la chaîne de compilation. Il est donc possible d'utiliser les compilateurs, soit de R. Pingali, soit de H.A.G Bik pour réaliser l'éparpillement. La sous-section 3.4.1 résume l'état de l'art en ce domaine.

3.4 Etat de l'art

Cette section décrit les travaux relatifs à la compilation appliquée aux programmes irréguliers. Deux thèmes se rapportent directement à nos travaux : la réécriture de programme dense en programme creux et la parallélisation dynamique de programmes irréguliers.

3.4.1 Réécriture de programmes denses en programmes creux

La motivation centrale de ces études est de simplifier la programmation des traitement sur les matrices creuses. Pour ce faire, le compilateur transformera un programme écrit sur des tableaux denses en un programme équivalent, mais écrit pour des structures creuses. De manière synthétique, le traitement de réécriture modélise en premier lieu l'exécution de l'algorithme afin d'abstraire son implantation, pour ensuite donner une nouvelle implantation utilisant une structure creuse. Le choix du modèle mathématique, sur lequel cette abstraction se fonde, est guidé par les caractéristiques d'une classe de structures creuses car il doit mettre en évidence, non seulement le comportement de l'algorithme sur la structure, mais aussi les optimisations possibles. Aussi, une distinction possible

Programme séquentiel initial.

do $\vec{I} \in D_I$
  P($\vec{I'}$))
  enddo
enddo

Programme parallélisé.

calculer le remplissage
calculer le graphe de dépendance creux
Θ = partition de D_I représentant un ordonnancement
do t = 1, $|\Theta|$
  doall $\vec{I} \in \Theta(t)$
    Sparse(P($\vec{I}$))
  enddo
  synchronisation
enddo

Le sequelette du programme séquentiel est représenté à gauche. Il décrit la manière le processus de parallélisation appréhende la structure d'un programme séquentiel.
- On considérera un ensemble de boucles imbriquées symbolisées par la notation vectorielle $\vec{I}$.
- Le corps de boucle est représenté par l'appel d'une fonction P. Il doit s'agir d'un programme à contrôle statique.

A droite figure le programme parallélisé selon un ordonnancement statique par front :
- la boucle d'indice t séquence les fronts.
- Chaque front $\Theta(t)$ se compose d'itérations à éxécuter en parallèle. La partition des itérations de D_I en front peut aussi s'effectuer en parallèle [SMC91] en appliquant un tri topologique du graphe de dépendances creux.
- Sparse(P($\vec{I'}$)) symbolise le programme réécrit en creux.

FIG. 3.8 – (prog.) Programme parallélisé

de ces travaux peut être établie en fonction de la classe de structures creuses choisie, auquelle correspond une modélisation spécifique.

Les travaux de A. J. C. Bik et H. A. G. Wijshoff [BW95], [BW96] fondent le traitement sur une généralisation du format C.C.S. ou C.R.S. La modélisation s'appuie sur l'algèbre des polyèdres.

V. Kotlyar, K. Pingali et P. Stoghill [KPS97], [Sto97] abordent ce problème en assimilant une matrice creuse à une base de données relationnelle. Cette approche considère une structure creuse comme une table associative où la clé d'accès est un indice. L'exécution du programme est alors formalisée dans l'algèbre relationnelle et le compilateur génère un programme utilisant des opérations sur les bases de données. Les auteurs résument leur stratégie de génération du programme creux comme étant «data centric» (centrée sur les données) ; le traitement est réécrit en fonction de l'énumération des entrées au lieu de l'énumération des itérations. Ces travaux sont complémentaires aux nôtres comme nous le montrons dans l'article [AD99]. ils peuvent s'intégrer dans notre schéma de parallélisation.

3.4.2 Parallélisation dynamique des programmes

La parallélisation dynamique calcule à l'exécution les dépendances et en déduit dynamiquement un ordonnancement. J. Saltz [SMC91] définit un schéma de parallélisation dynamique des programmes irréguliers selon le mode inspecteur-exécuteur. L'inspecteur calcule le graphe de dépendances et effectue un ordonnancement des itérations. L'exécuteur calcule en parallèle le programme numérique. Les travaux que nous présentons suivent un schéma inspecteur-exécuteur dont les fonctions sont similaires. Toutefois, les hypothèses initiales sont différentes. Dans les travaux de J. Saltz, l'analyse de dépendances s'effectue selon les conditions de Bernstein sur des programmes possédant des indirections et en utilisant le résultat du remplissage. Dans nos travaux, l'analyse de dépendances s'effectue à partir d'un code dense en raffinant les conditions de Bernstein et en intégrant le remplissage. Comparativement aux travaux de J. Saltz, la contribution que nous apportons réside dans l'intégration du calcul du remplissage et dans le mode de calcul des dépendances. En effet, avec les termes utilisés dans cet article, l'analyse des dépendances selon la méthode de J. Saltz s'effectue dynamiquement selon les conditions de Bernstein en vérifiant toutefois que l'accès commun à une cellule mémoire correspond à une entrée. Nous la désignerons sous le terme d'analyse CBRE (analyse selon les Conditions de Bernstein Restreintes aux Entrées). Des expériences [AD99] comparent les ordonnancements obtenus à partir de graphes de dépendances calculés montre un gain effectif de l'analyse creuse par rapport à l'analyse CBRE. Qualitativement, ce gain se justifie par la prise en compte de la propriété d'absorption pour le calcul des dépendances qui réduit le nombre de dépendances.

La bibliothèque RAPID [FY97] est un support d'exécution pour l'ordonnancement des graphes acycliques irréguliers. Le graphe de dépendances est déduit d'une description faite par l'utilisateur des objets manipulés, des tâches et des accès à ces tâches. Nos travaux offrent une automatisation de l'extraction du graphe de dépendances et un calcul symbolique du remplissage qui suppose avoir été déjà réalisé par une méthode ad-hoc. Pour la phase d'ordonnancement, les stratégies d'ordonnancement développées dans les travaux cités peuvent être adaptées au cadre de l'étude que nous proposons.

3.5 Bilan

Cette méthode utilisée pour la parallélisation de la factorisation de Cholesky creuse [HNP91], échappait à la fois au cadre d'analyse des compilateurs parallélisant, et à la définition classique des dépendances déterminées selon les conditions de Bernstein. L'étude réalisée pour paralléliser les structures creuses se fonde sur une analyse de dépendances élargissant la notion de dépendances pour permettre d'évaluer les cellules dont la valeur était modifiée par le calcul. La source de ce parallélisme se fondait à la fois sur la propriété d'absorption et de neutralité ; propriétés qui se trouvent unifiées par la notion de non-substitution. Cette propriété permet de donner une formalisation cohérente avec la définition classique des dépendances en la complétant par des conditions portant sur les valeurs des données. Nous avons proposé une chaîne de parallélisation automatique pour le creux en validant expérimentalement certains éléments : un noyau d'analyseur statique pour le creux développé en Ocaml et utilisant la librairie OMEGA [Pug92] et une librairie de structures creuses génériques développée en Open MP. Des validations manuelles utilisant cette librairie ont été réalisés sur différents problèmes.

Ce travail a été développé avec R. Adle dans le cadre de DEA puis de sa thèse [Adl99] et nous avons collaboré avec M.Aiguier [AAD00]. Il s'inscrit dans l'action HIPERF du GDR ARP. En parallèle à l'extension des dépendances aux dépendances creuses, cette approche s'appuie sur une utilisation originale de la conception d'inspecteur-exécuteur. L'inspecteur se charge de réaliser les calculs relatifs à l'interprétation abstraite permettant d'obtenir le graphe de dépendances sur lequel se détermine l'ordonnancement des tâches. Contrairement à la définition classique d'inspecteur qui correspond à un programme générique, cette approche définit un inspecteur particulier à chaque programme. Plus précisemment, la partie particulière à chaque programme concerne les formules permettant de calculer le remplissage des données et les dépendances. Le cadre commun à tous les inspecteurs concerne l'application de ces formules.

Chapitre 4

PARADEIS

4.1 Introduction

Les recherches menées en amont de la chaîne de compilation, sur la transformation automatique de code ouvrait à une recherche en aval, sur la définition de librairie adaptée aux structures creuses afin de fournir des *« back-end »* adaptés à la génération du code creux sur les grappes de PCs. La solution proposée pour répondre à cette question étend ce cadre plus largement en fournissant une interface de programmmation creuse selon le principe du parallélisme de données.

Dans cette optique, PARADEIS supporte une programmation SPMD et simule un espace d'adressage unique pour des architectures à mémoire distribuée ou des réseaux de stations de travail. Ce cadre s'approche de la programmation séquentielle ; il a pour but final de rendre la programmation des applications parallèles pour le traitement des matrices creuses aussi simple que la programmation séquentielle avec des tableaux denses.

Offrir une abstraction suffisante permettant de rendre simple la programmation parallèle tout en étant immédiatement portable constitue la motivation qui guide le développement de la bibliothèque PARADEIS dans un langage orienté objet s'appuyant sur la bibliothèque STL [SL94]. Le langage C++ [Str99] supporte l'encapsulation et la généricité des motifs (*templates*) ce qui permet à la bibliothèque STL de proposer une interface entre les données et les algorithmes qui s'y appliquent. Cette approche permet de simplifier la réalisation de programme et offre un cadre de programmation modulaire où le programme est plus indépendant de la structure de donnée. PARADEIS est un environnement de programmation parallèle implanté en C++. Le modèle de programmation en PARADEIS se base sur le modèle BSP pour l'aspect de la conception de programme parallèle et sur les concepts de la bibliothèque STL pour la programmation des structures creuses en parallèle.

Le modèle BSP a pour but de proposer un standard qui soit à la fois un modèle d'exécution sur les architectures parallèles, de programmation des langages de haut niveau et de complexité d'un calcul parallèle. Selon ces auteurs [Val90], un tel lien permet d'avoir l'assurance qu'un programme écrit sur ce modèle s'exécutera de manière optimale sur une architecture inspirée du même modèle comme c'est le cas pour le modèle séquentiel.

BSP modélise 3 composants classiques des algorithmes parallèles :
- les éléments de calcul ou de mémorisation,
- les éléments de communication pour échanger des messages entre éléments de calculs ou de mémorisation,
- la possibilité de synchroniser tout ou partie de ces éléments de calculs ou de mémorisation simultanément et ainsi de passer à l'étape de calcul suivante si tous ces éléments ont terminé leur traitement.

La phase de synchronisation peut être réalisée lors d'une phase de communication globale entre tous les processeurs. Sur chaque processeur, cette communication a lieu lorsque le calcul précédent est terminé. Elle a donc lieu lorsque tous les processeurs impliqués ont terminé la phase de calcul précédente à laquelle succède une nouvelle phase de calcul. L'application de ce modèle aux algorithmes conduit donc à des programmes alternant des séquences de calculs parallèles et des phases de communications. Dans ce cadre, PARADEIS offre un langage où les algorithmes BSP s'expriment de façon naturelle. Les primitives de communications sont systématiquement exprimées comme des primitives de communication généralisées, fondées sur des références à un espace d'adressage unique. Nous renvoyons le lecteur aux articles [RD00b, RD00a] pour un développement plus approfondi de ces notions.

La bibliothèque STL est basée principalement sur trois composants : les *containers*, les itérateurs et les algorithmes. Les conteneurs (*containers*) sont des objets utilisés pour stocker d'autres objets. Les itérateurs représentent les méthodes de parcours des éléments des conteneurs. Les algorithmes se définissent sur les conteneurs ainsi que les itérateurs proposés. La bibliothèque STL définit trois classes d'itérateurs. L'itérateur *forward* impose un accès séquentiel unidirectionnel. L'itérateur *bidirectional* permet un accès séquentiel bidirectionnel. Finalement, l'itérateur *random* permet un accès aléatoire. Ces itérateurs constituent l'interface entre la structure de donnée et l'algorithme. Les extensions proposées de la STL se concentrent sur le traitement des matrices creuses en parallèle.

Il existe de nombreux formats de stockage spécialisés pour les structures creuses. Chaque format est adapté à certains critères de manipulation de ces structures que ce soit une distribution ou un type de parcours particulier (par exemple le parcours pour la multiplication matrice vecteur). L'utilisation d'un langage orienté objet permet d'unifier les différentes interfaces proposées par tous ces formats de stockage. Le concept de collection dans les langages orientés objets introduit des classes génériques représentant un ensemble d'objets et fournissant une interface pour les manipuler. Les collections peuvent être vues comme une classe mère *Collection* et ses spécialisations en *Ensemble*, *Liste*, *Tableau*, etc. Ainsi toutes les méthodes (union, recherche, accès à un élément, ...) qui sont définies pour *Collection*, s'appliquent aux classes qui en héritent. La bibliothèque STL reprend les concepts de collection. Bien que basée sur les concepts objets, en particulier l'encapsulation des données et des méthodes associées, la STL ne respecte pas l'association entre les données et certaines méthodes, les méthodes de parcours des structures. Cette approche convient bien à la définition de structures spécialisées et de méthodes de parcours comme dans le domaine des structures creuses.

Les sections suivantes exposent le modèle de programmation de PARADEIS qui permet d'implanter des algorithmes data-parallèle indépendamment de la distribution initiale des données. La contribution de PARADEIS est de définir des extensions qui portent sur la définition d'un nouveau

```
typedef vector<int> MyArray ;

MyArray R, V1, V2 ;

MyArray : :iterator i, i1, i2 ;
for(i = R.begin(), i1 = V1.begin(), i2 = V2.begin() ;
    i1 != V1.end() && i2 != V2.end() ;
    i.next(), i1.next(), i2.next())
   (*i) = (*i1) + (*i2) ;
```

FIG. 4.1 – (prog.) Somme de vecteurs denses en STL : R = V1 + V2

conteneur et d'itérateurs associés permettant d'exprimer des calculs parallèles sur des structures creuses.

4.2 Composants logiciels de PARADEIS

PARADEIS est une bibliothèque regroupant les éléments nécessaires à l'expression simple de calcul parallèle sur des structures creuses distribuées. Afin de cerner son cadre d'utilisation, nous allons étudier un programme introductif au modèle de développement de PARADEIS, l'addition de vecteurs creux. Il s'agit d'un exemple simple qui met en œuvre des mécanismes de calcul essentiels en PARADEIS. On constatera, en particulier, que l'expression d'un programme parallèle opérant sur des vecteurs creux est tout à fait similaire à celui fait en séquentiel pour des vecteurs denses STL, bien que celui-ci soit réalisé pour des structures creuses et en parallèle. La figure 4.1 montre une façon d'exprimer le calcul de la somme de vecteur en STL. Les deux points importants à noter ici sont l'existence de trois itérateurs i, $i1$ et $i2$ effectuant le parcours des trois structures R, $V1$ et $V2$ et le test de fin de boucle exprimé ici sur le plus petit des vecteurs $V1$ ou $V2$. Le programme 4.2 est similaire au précédent mais sur des vecteurs creux distribués. Les vecteurs à sommer $V1$ et $V2$ sont supposés être déclarés avant le vecteur résultat R. La déclaration des éléments d'itération suit la déclaration de ces structures. Pour finir, la surcharge d'opérateurs permet d'exprimer la somme de vecteurs de manière habituelle comme sur des tableaux.

Nous montrerons dans les sous-sections suivantes les solutions proposées pour permettre la rédaction d'un programme maintenant un flot de contrôle unique, utilisant un espace d'adressage unique et masquant l'utilisation du creux. Le détail des mécanismes de PARADEIS est introduit par les sous-sections suivantes : la sous-section 4.2.1 aborde la gestion des structures creuses. La sous-section 4.2.2 montre les itérateurs associés à la structure creuse introduite dans PARADEIS. La sous section 4.2.3 présente les contraintes liées à l'expression des calculs creux et la sous-section 4.2.4 expose l'impact des communications sur les calculs.

4.2.1 Structure creuse

PARADEIS reprend les concepts de la STL pour les étendre aux traitements des structures creuses parallèles. Le conteneur `SparseArray` représente une structure creuse distribuée.

SparseArray en PARADEIS

```
SparseArray M(100, 100) ; // matrice 100×100
SparseArray V(100) ;      // vecteur (1 dimension)
M.partiBRD(fichier) ;     // partitionnement BRD à partir
// d'un fichier contenant la matrice creuse au format MTX
```

Une matrice creuse correspond à un tableau dont les zéros ont été éliminés [DER86]. Ces structures doivent répondre à deux exigences : la première est de compresser la représentation d'un tableau afin de ne conserver que les valeurs différentes de zéro, la seconde est de conserver la correspondance des valeurs avec les indices du tableau compressé afin de pouvoir combiner des calculs entre matrices creuses et matrices denses ou encore entre différentes matrices creuses. Ces deux exigences entraînent des références avec des indirections qui rendent complexe la programmation en parallèle des programmes manipulant les structures creuses [PHS+95]. Pour simplifier la programmation, nous adoptons une solution qui est de rendre le creux implicite. Pour l'utilisateur, la matrice creuse est vue comme un tableau dense sur lequel des opérations couvrant tous les indices denses peuvent être appliquées. Cependant, les calculs s'opèrent uniquement sur les valeurs de la matrice pour lesquelles une valeur différente de zéro est produite, que cette matrice soit le résultat du calcul ou une variable temporaire nécessaire à celui-ci.

Le premier point à prendre en compte dans le programme 4.2 est l'impact de la représentation creuse des structures sur leur manipulation. Dans le cas de l'addition de vecteurs, le calcul des valeurs de R se restreint alors aux indices pour lesquels $V1$ ou $V2$ possède une valeur non nulle. Contrairement à un programme de somme de vecteurs denses, la structure de R, vecteur creux résultat, dépend des vecteurs $V1$ et $V2$. La structure de R sera définie par l'ensemble des entrées (index des valeurs non nulles) E_R : $E_R = \{i | i \in E_{V1} \vee i \in E_{V2}\}$ soit $E_R = E_{V1} \cup E_{V2}$. Pour définir cette union des entrées, PARADEIS contient un ensemble d'opérations ensembliste agissant sur les entrées. Nous définissons plus longuement ces opérations qui permettent de définir la création de structure dans la section 4.3.2. Par exemple, l'union, dans la déclaration `SparseArray R(union(V1, V2))`, définit la structure de R, c'est-à-dire l'ensemble des valeurs qui aboutirons au calcul d'une somme non nulle.

La manipulation d'une structure creuse (figure 3.1) requiert d'expliciter le traitement sur la structure compressée. L'algorithme doit intégrer la représentation creuse. Dans le programme 3.1, les bornes de boucles sont des tableaux décrivant le format de stockage creux CRS (Compressed Row Storage). Aussi, bien que nécessaire à la réduction de l'occupation mémoire, l'utilisation des structures compressées rend le programme plus complexe. Afin de simplifier le traitement strictement lié à la structure, PARADEIS masque l'utilisation des formats creux. Un format creux est alors manipulé comme les structures STL génériques, au travers d'itérateurs.

4.2.2 Itérateurs

Le principe des itérateurs est de fournir une méthode d'accès aux structures de données basée sur une classification des types de parcours. PARADEIS dispose donc d'itérateurs adaptés au conteneur représentant les structures creuses nommé `SparseArray`. Les itérateurs introduit dans PARADEIS sont des itérateurs dits creux car ils parcourent les éléments du conteneur `SparseArray` qui représente une matrice creuse. Les classes d'itérateurs sont les itérateurs parallèles et les itérateurs séquentiels. Les points communs entre ces 2 types d'itérateurs, développés dans les sections suivantes, sont premièrement qu'ils ne parcourent que les valeurs stockées et donc pas les zéros de la matrices et deuxièmement, qu'ils opèrent sur la structure distribuée.

```
SparseArray R(union(V1, V2)) ;
Iterator i(R), i1(V1,i), i2(V2,i) ;

V1.exchange(R, Block(Section(1,100)),
               Block(Section(1,100))) ;
V2.exchange(R, Block(Section(1,100)),
               Block(Section(1,100))) ;
for(i.begin(), i1.begin(), i2.begin() ;
    i.end() ;
    i.next(), i1.next(), i2.next())
   R[i] = V1[i1] + V2[i2] ;
```

FIG. 4.2 – (prog.) Somme de vecteur en parallèle avec communications

4.2.3 Synchronisation

Deux éléments composent un `SparseArray` : sa structure et son espace de valeurs. Sa structure se présente sous la forme d'un ensemble d'indices des valeurs non nulles, appelées les entrées. Ainsi, ne parcourir que les valeurs significatives signifie que le calcul s'accompli uniquement sur les entrées. En particulier, dans la somme de vecteur creux `R[i] = V1[i1] + V2[i2]`, l'itérateur i parcours uniquement les entrées de R. Cependant, les indices i, $i1$ et $i2$ doivent posséder la même valeur à chaque étape. Il faut introduire un mécanisme de mise en correspondance, entre les structures manipulées dans une même expression, que nous nommons la « synchronisation ».

Ce phénomène est visible sur les formats de compression classique de matrices creuses. La figure 3.1 montre une matrice creuse et sa représentation dans le format compressé CRS. Le principe est donc de prendre un itérateur comme référence et de synchroniser les autres sur celui-ci. Cette synchronisation opère sur l'espace du résultat car c'est cet espace qui détermine quels sont les éléments qui seront différents de zéro en fin de calcul. Pour effectuer une telle opération, l'espace des entrées à été précédemment défini par un calcul symbolique comme l'union des entrées de $V1$ et $V2$ (c.f. 4.3.2). La synchronisation du parcours de $V1$ et $V2$ sur R, s'exprime au niveau de la déclaration des itérateurs : `Iterator i(R), i1(V1,i) ;` ou i parcours R et $i1$ parcours $V1$ en se synchronisant sur i.

4.2.4 Relation calcul-communication

Les deux principaux éléments constitutifs de PARADEIS sont la structure creuse distribuée et les itérateurs parallèles qui lui sont associés. L'aspect data-parallèle dans PARADEIS introduit des itérateurs parallèles qui parcourent sur chaque processeur les valeurs stockées et ne nécessite donc pas de communication pendant ce calcul. Cette indépendance entre les phases de communications et de calculs offre une grande liberté dans l'expression de calcul parallèle mais conduit à décrire explicitement les échanges de données nécessaires à chaque calcul.

Aussi, si le placement initial n'est pas approprié au calcul, il est nécessaire d'effectuer une communication pour obtenir un placement adéquat au calcul. Dans le programme de la figure 4.2, la primitive `V1.exchange(R, ...)` spécifie que les données de $V1$ doivent être échangées afin que les entrées présentes pour R et $V1$ coïncident. La section 4.3.2 détaille la sémantique de cet échange.

4.2.5 Résumé

La bibliothèque STL crée des objets représentant des structures et d'autres représentant des algorithmes afin de mieux les séparer offrant ainsi, une grande flexibilité dans la programmation. Un des avantages de cette technique est de pouvoir implanter un algorithme indépendamment de la structure de donnée sous-jacente. L'algorithme ne dépend que des schémas de parcours de structure. Toute structure disposant de tels schémas peut être employée avec cet algorithme. Cette indépendance entre traitement et structure de donnée est obtenue au travers du mécanisme de généricité des motifs du langage C++. Un tel mécanisme a également été proposé pour le langage Java avec la librairie JGL [Obj97]. PARADEIS permet d'exprimer des calculs parallèles sur des structures creuses dans l'esprit des conteneurs et des itérateurs de la bibliothèque STL. Afin de prendre en compte les particularités du calcul creux sur ces structures, PARADEIS dispose d'opérateurs spécifiques.

4.3 Structure et communication

Cette section décrit les éléments principaux d'implantion de PARADEIS. Fournir une structure distribuée avec les fonctionnalités mentionnés conduit à résoudre trois problèmes principaux : Implanter une structure creuse distribuée, Gérer les communications implicites exprimées sous forme d'accès mémoire et Gérer le placement des données par rapport à l'irrégularité. Dans cette section nous aborderons les deux premiers problèmes. La section suivante traite du dernier dans la mesure où celui-ci définit un problème de placement qui présente une indépendance thématique vis-à-vis des premiers problèmes car son cadre applicatif est plus large que celui des matrices creuses. Toutefois, les algorithmes et les analyses théoriques ont été faites pour le placement de la structure creuse parallèle.

4.3.1 Descripteur

L'ensemble des contraintes imposées pour l'expression des communications et la simulation d'un espace d'adressage unique dans PARADEIS conduit à la réalisation d'une structure spécialisée qui permet de représenter la distribution d'une structure creuse.

Afin d'optimiser les communications creuses, nous considérons l'organisation du descripteur suivante (figure 4.3) :

- la racine donne la taille de la matrice,
- le premier niveau représente l'information globale partagée par tous les processeurs. C'est une approximation du placement,
- le second niveau correspond à la connaissance locale qui est une description exacte des données que le processeur stocke,
- les feuilles contiennent les données présentes sur le processeur. Elles se composent de tableaux denses.

Cette organisation est inspirée de celle de la structure de donnée R-Tree [Gut84] utilisée dans les bases de données géographiques. Cette structure répond au problème de la localisation d'éléments en fonction de leurs coordonnées. Dans le cas d'éléments consécutifs (géographiquement) il est alors possible de chercher un segments de coordonnées.

La particularité de ce descripteur est de définir une description hiérarchique du partitionnement. Chaque processeur possède ainsi une approximation de ce partitionnement. Cette approximation est conservative dans le sens où tous les indices des valeurs présentes sont inclus dans la description du

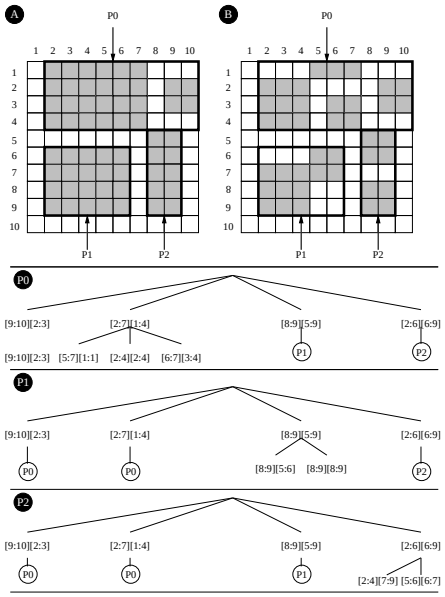


FIG. 4.3 – Descripteur d'une matrice 10×10 (racine, premier et second niveau)

niveau supérieur. L'intérêt de cette hiérarchie est de pouvoir exprimer le partitionnement en tenant compte des contraintes mémoire. En effet, la description du niveau global présente sur tous les processeurs se définit avec moins d'éléments que celle du niveau local. La précision de la description du niveau globale peut être ainsi modulée en fonction de l'espace mémoire disponible.

Le rôle du descripteur est donc de représenter le placement des données distribuées sur l'ensemble des processeurs. Le descripteur au niveau global est composé de blocs qui définissent ce placement. La représentation en blocs convient pour l'ensemble des problèmes de l'algèbre linéaire creuse. En effet les schémas de communication les plus efficaces associés en particulier au produit matrice vecteur dépendent d'un placement de la matrice en blocs de colonne ou de ligne. Cependant la description du `SparseArray`n'est pas limitée à ce type de partitionnement et tout découpage en blocs qui respecte la condition que le niveau global est une partition convient.

4.3.2 compilation des communications pour les structures creuses

Dans PARADEIS, les choix qui ont conduit au développement de la structure creuse pour sa construction et son utilisation, sont guidés par la possibilité de calculer les ensembles Send, Receive et Compute (Section 2) sur chaque processeur. Cela permet d'avoir une unique communication entre chaque processeur dans le cas d'une référence globale à la structure. Les deux principes qui conduisent à ce résultat sont l'utilisation du descripteur hiérarchique présenté dans la section 4.3.1 et l'introduction dans les applications d'une structure creuse dont la programmation rend le creux implicite. Rappelons ici que les références aux tableaux s'effectuent de la même manière que si elles étaient destinées au calcul avec un tableau dense.

Ces choix ont des conséquences sur le schéma inspecteur-exécuteur proposé dans la section **??**. Le calcul de l'ensemble Compute est immédiat puisque chaque processeur connaît les données qu'il

stocke et que les accès aux structures creuses sont fait comme sur des structures denses, c'est-à-dire par la même fonction affine. La section suivante montre le calcul des ensembles Send et Recv dans ce contexte.

Inspecteur-exécuteur

Le descripteur associé à un schéma inspecteur-exécuteur permet de calculer les communications exprimées par des sections de tableaux. La compilation des communications est effectuée conformément à l'expression d'une affectation en HPF :

`X(`$l_x : u_x : s_x$`) = Y(`$l_y : u_y : s_y$`)`

ou `X` et `Y` sont considérés comme creux, $l_x, u_x, s_x, l_y, u_y, s_y$ sont des vecteurs d'entiers représentant des sections à plusieurs dimensions, où l_x (resp. l_y) est le début de la section, u_x (resp. u_y) la fin et s_x (resp. s_y) le pas. Dans la bibliothèque PARADEIS, la communication associée est réalisée par l'appel de la méthode `Y.exchange(X,` $l_x : u_x : s_x$`,` $l_y : u_y : s_y$`)`. Cette méthode ne réalise que l'échange de données nécessaire à l'affectation mais n'effectue pas l'affectation en elle-même.

Le schéma de communication est fondé sur l'extraction des données significatives appartenant à une section de tableau en utilisant un arbre dont les niveaux sont ordonnés par l'inclusion. Le schéma inspecteur-exécuteur se décompose en deux phases :

Approximation Tout d'abord, la recherche est effectuée sur le premier niveau de l'arbre. Le résultat est une approximation de l'ensemble des valeurs significatives appartenant au processeur.

Raffinement Cette recherche est accomplie sur le second niveau par chaque processeur, pour envoyer les valeurs non nulles qu'il possède.

Notations : Nous appelons *bloc creux*, l'union de sections de tableaux à plusieurs dimensions. Un *bloc creux global* est un sous-ensemble de blocs décrit au niveau global, tandis qu'un *bloc creux local* est un sous-ensemble de blocs creux locaux. GS_p^X et LS_p^X correspondent respectivement à la description globale et locale de X sur le processeur p. Par extension, GS_*^X représente l'union de tous les blocs creux globaux de tous les processeurs. Soit X un tableau creux, nous définissons les fonctions suivantes :

- $Own(X, s) = (P, B)$ donne l'ensemble P des processeurs possédant une partie du bloc creux s d'après GS_*^X. B est un ensemble de blocs creux dans lequel chaque élément est indexé par un identificateur de processeur tel que pour chaque $p \in P$, son bloc creux correspondant est $B_p \in B$.
- $own(X, s) = P$ représente la restriction de la fonction précédente Own à l'ensemble des processeurs.
- $\sigma(a, b, c)$ est une fonction qui donne un sous-ensemble de blocs $c' \subseteq a$ correspondant à $c \subseteq b$. Cette fonction calcule la correspondance entre les sections écrites et les sections lues.

  1. Si a, b, c sont des sections de tableau unidimensionnel telles que :
     $a = [l_a : u_a : s_a]$, $b = [l_b : u_b : s_b]$, $c = [l_c : u_c : s_c]$, alors la section résultante est :

     $$c' = [l_a + s_a \frac{l_c - l_b}{s_b} : l_a + s_a . \lfloor \frac{u_c - l_b}{s_b} \rfloor : \frac{s_c}{s_b} s_a]$$

  2. Si a, b, c sont des sections à plusieurs dimensions alors σ correspond à l'application de la formule précédente sur chacune des dimensions.

3. Si a, b, c sont des blocs creux, alors le résultat est l'union des sections de tableau élémentaires calculé à partir de la formule ci-dessus.

La partie recevant l'affectation dans un tableau peut être déterminée pour une sous section donnée en utilisant la fonction σ. Pour l'affectation

$$A[0:21:3] = B[10:38:4]$$

la partie affectée correspondant à `B[18 :34 :8]` est :

$$\sigma([0:21:3],[10:38:4],[18:34:8]) = [6:18:6]$$

Alors `A[6 :18 :6]` recevra les valeurs de `B[18 :34 :8]`.

La figure 4.4 montre la procédure d'extraction d'une sous-section (application de la fonction σ) du tableau A à partir d'une sous-section de B. L'étape 1 montre l'ensemble des indices présents dans le tableau B. A l'étape 2, l'ensemble B' est la restriction de B aux bornes [18 :34] c'est-à-dire du troisième au septième élément de B. L'étape 3 applique le pas de 8 soit un élément sur 2. Les étapes 4 à 6 reprennent les étapes précédentes en les appliquant au tableau A pour obtenir le tableau $A_{écrit}$, symétrique du tableau B_{lu}, qui sont tous deux les sections effectivement lues et écrites.

étapes	Ensemble	Section	Indices
1	B	[10 :38 :4]	10 14 [18 22 26 30 34] 38
2	B'	[18 :34]	[18 22 26 30 34]
3	B_{lu}	[18 :34 :8]	[18 26 34]
4	A	[0 :21 :3]	0 3 [6 9 12 15 18] 21
5	A'	[6 :18]	[6 9 12 15 18]
6	$A_{écrit}$	[6 :18 :6]	[6 12 18]

FIG. 4.4 – Exemple de correspondance entre sections lues et écrites (fonction σ)

Phase inspecteur

La première étape est le calcul des ensembles *Send* et *Receive*. Ils sont définis à partir des informations globales. Dans la suite *me* dénote l'identificateur du processeur.

Inspecteur en émission Son but est de définir une approximation des blocs qui sont envoyés par le processeur *me*. L'inspecteur compare les informations globales et la section de tableau pour obtenir l'ensemble des processeurs récepteurs et les données à émettre.

Approximation

- GR est l'ensemble des blocs de `X` au niveau global qui correspondent aux données de `Y` appartenant au processeur *me* :

$$GR = \sigma([l_x : u_x : s_x], [l_y : u_y : s_y], GS^Y_{me} \cap [l_y : u_y : s_y])$$

- (PS, B) est l'ensemble des processeurs qui reçoivent les valeurs de X appartenant à *me*. B est l'ensemble correspondant de blocs du niveau global :

$$(PS, B) = Own(X, GR)$$

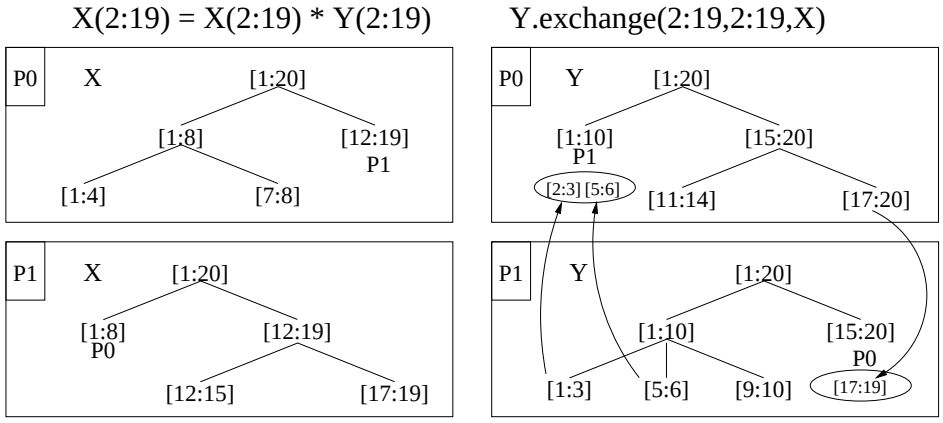


FIG. 4.5 – Exemple d'exchange avec le descripteur hiérarchique

Raffinement

– RB_p est l'ensemble de blocs de Y au niveau local qui seront envoyés au processeur p :

$$\forall p \in PS,\ RB_p = LS^Y_{me} \cap \sigma([l_y : u_y : s_y], [l_x : u_x : s_x], B)$$

Inspecteur en réception L'inspecteur pour la réception calcule l'ensemble des processeurs qui lui envoient des données. L'envoi et la réception étant déterminés à partir de la même information qui est dupliquée, on ne peut se trouver ni en situation d'un message non émis, ni en situation d'émission d'un message sans destinataire.

– GW représente l'ensemble des adresses de `X` qui nécessitent une communication pour leur affectation :

$$GW = \sigma([l_y : u_y : s_y], [l_x : u_x : s_x], GS^X_{me} \cap [l_x : u_x : s_x])$$

– PR est l'ensemble des processeurs qui sont supposés envoyer des données :

$$PR = own(Y, GW)$$

Phase exécuteur

Seules les valeurs significatives sont transmises pendant la communication. Par conséquent, la phase inspecteur ne transmet que le descripteur local et les valeurs associées. Ces valeurs sont stockées dans une partie temporaire dédiée de `Y`.

Exécuteur émission

```
for p ∈ PS − {me}
    send(RB_p,value(RB_p),tag,p)
endfor
```

Exécuteur réception

```
for p ∈ PR − {me}
    receive(B,D,tag,p)
    store (p,B,D) into LS^Y
endfor
```

P_0	P_1
• Calcul de GR, les parties de X pour lesquelles P_{me} possède Y	
$GR = \sigma([2:19]_X, [2:19]_Y, GS^Y_{me} \cap [2:19]_Y)$	
$GS^Y_{me} = [15:20]_Y$	$GS^Y_{me} = [1:10]_Y$
$GR = [15:19]_X$	$GR = [2:10]_X$
• Listes des PE qui possèdent une partie de X qui recevra un Y que je possède	
$(PS, B) = Owm(X, GR)$	
$(PS, B) = (P_1, [15:19]_X)$	$(PS, B) = (P_0, [2:8]_X)$
• Les parties de Y qui sont envoyées à chaque PE	
$\forall p \in PS,\ RB_p = LS^Y_{me} \cap \sigma([2:19]_Y, [2:19]_X, B)$	
$LS^Y_{me} = \{[11:14]_Y, [17:20]_Y\}$	$LS^Y_{me} = \{[1:3]_Y, [5:6]_Y, [9:10]_Y\}$
$RB_1 = LS^Y_0 \cap [15:19]_Y$	$RB_0 = LS^Y_1 \cap [2:8]_Y$
$RB_1 = [17:19]_Y$	$RB_0 = \{[2:3]_Y, [5:6]_Y\}$
• Réciproque de GR, les parties de Y pour lesquelles P_{me} possède X	
$GW = \sigma([2:19]_Y, [2:19]_X, GS^X_{me} \cap [2:19]_X)$	
$GS^X_0 = [1:8]_X$	$GS^X_1 = [112:19]_X$
$GW = [1:8]_Y$	$GW = [12:19]_Y$
• Liste des PE qui stockent une partie de Y qui me sera envoyée	
$PR = own(Y, GW)$	
$PR = P_1$	$PR = P_0$

TAB. 4.1 – Exemple de calcul des ensembles Send et Recv dans l'exchange

Sur l'exemple de la figure 4.5, les communications nécessaires au calcul de X[2 :19] = X[2 :19] * Y[2 :19] correspondent à l'appel de la fonction Y.exchange(X, [2 :19], [2 :19]). La table 4.1 reprend les étapes de calcul des ensembles Send et Recv qui sont utilisés dans la phase exécuteur. Dans cette table, les sections sont annotées par la structure X ou Y à laquelle elles appartiennent. Dans cet exemple, les détails du calcul de la fonction σ sont ignorés car la plage d'échange est identique pour X et Y. En particulier pour deux tableaux quelconques A et B, $\forall(u,l), \sigma([2:19]_A, [2:19]_B, [u:l]_B) = [u:l]_A$.

PARADEIS utilise, pour simuler un espace d'adressage unique, un support exécutif de type inspecteur-exécuteur. La représentation du partitionnement est donnée par un descripteur dédié à la gestion de l'irrégularité.

En effet, en considérant l'espace creux comme un sous-ensemble de l'espace dense, nous définissons les échanges sur cet espace dense. Ceci permet de les exprimer par des expressions de tableaux, autorisant ainsi l'emploi d'optimisations de compilation des communications pour le traitement des

programmes réguliers [FC96]. Plus précisément, toutes les données destinées à un même récepteur sont envoyées dans un unique message, c'est l'optimisation dite de vectorisation de messages.

Opérations de traitement symbolique

En suivant la règle du « le propriétaire calcule », les instructions du programme prennent comme référence l'espace d'itération de la structure en écriture. La définition des espaces sur lesquels un calcul doit s'opérer, s'effectue par rapport à la matrice en écriture. Le calcul symbolique permettant de définir l'espace contenant les valeurs significatives de la matrice résultat est défini par des opérations ensemblistes.

Les communications et les calculs opèrent en fonction de la structure creuse de la matrice, c'est-à-dire en fonction des indices du tableau pour lesquels la valeur associée est différente de zéro. Dans la version actuelle de PARADEIS, la déclaration de cette structure est statique. Une fois la structure établie, elle n'évolue plus.

Afin d'établir une structure qui corresponde à la représentation d'une matrice après son calcul, PARADEIS offre des primitives appliquées aux structures qui permettent de définir par un calcul symbolique cette représentation (cf. Chapitre 2). La factorisation creuse de Cholesky [HNP91] est un exemple typique où cette technique s'applique. Les opérations fournies par PARADEIS pour ces calculs symboliques sont :

- opérations ensemblistes :
  - union : `union()`
  - intersection : `inter()`
  - création avec copie : `Copy()`
- opération d'extraction de structure par ligne et colonne : `extracthyp()`
- opération logique : filtre par un prédicat : `select()`

Permutation sans communication

Dans le domaine des communications exprimées par l'échange de données entre structures, certaines expressions dénotent un échange d'une structure sur elle-même. Ces communications sont hors du cadre de la primitive `exchange` car elles impliquent une modification de la représentation de la structure. C'est le cas dans l'exemple du décalage exprimé par le calcul suivant : $B[2:101] = B[1:100]$. La structure `B` n'est plus définie sur le même intervalle ce qui se traduit par une modification de son descripteur. Dans le cadre des placements dis réguliers, ce type de transformation se traduit généralement par une communication des données se trouvant à la frontière entre processeurs. Dans cette partie, nous allons nous intéresser aux transformations qui grâce à notre structure de données hiérarchique peuvent être effectuées sans communication.

En effet, contrairement aux tableaux dont l'indice représente une position physique en mémoire, les formats de stockage creux effectuent uniquement une association logique entre indices et valeurs. Cette association permet de mettre en relation les valeurs lors du calcul en se fondant sur la représentation dense. Cette caractéristique des stockages creux que nous reprenons pour le descripteur permet d'appliquer des permutations aux structures creuses sans communication. Pour cela, il suffit de modifier localement la relation entre indice et valeur. L'exemple du décalage $B[2:101] = B[1:100]$, ne produit pas de communication mais seulement l'incrémentation des valeurs de début et de fin de chaque section de tableau du descripteur.

Ces modifications peuvent être modélisées par une fonction qui transforme les coordonnées des points de la matrice. Pour que l'image du descripteur par cette fonction soit un descripteur, nous imposons deux conditions. La première découle de la définition d'un descripteur à savoir qu'il représente un partitionnement de la matrice. La seconde, pour des raisons d'implantation, est que cette fonction s'exprime sur chaque section en transformant les bornes des sections constituant les blocs du descripteur.

4.4 Partitionnement

La construction du descripteur d'une matrice creuse distribuée organise l'espace des données en une partition de blocs. Ce format amène à représenter des valeurs nulles habituellement éliminées afin de simplifier la trame des blocs ainsi créés. Ils englobent l'ensemble des points non nuls et des valeurs nulles voisines. Le nombre de blocs (noté K) et de valeurs nulles stockés (noté Q) constituent alors les paramètres quantitatifs de ce descripteur à partir desquels on détermine l'occupation mémoire M du descripteur et d'une plus complexe le temps d'exécution. La tentation d'augmenter la précision de la description pour réduire le nombre de traitements effectué sur les zéros des blocs, doit être modérée par le surcoût qu'introduit une augmentation du nombre de blocs. Inversement celle de réduire le nombre de blocs est contre-balancé par l'augmentation des valeurs nulles du à l'accroissemnt de l'aire des blocs regroupant les données significatives. On se retrouve donc face à un problème d'optimisation du descripteur en regard de la minimisation du temps d'exécution des programmes et de manière plus secondaire, l'ocupation mémoire. Ce problème, que nous nommons $SBCP$, détermine une partition de la matrice en K blocs contenant au plus Q données. Nous avons prouvé qu'il était $\mathcal{NP}$-complet. Nous avons aussi montré que le problème demeure $\mathcal{NP}$-complet si l'on remplace la partition par une couverture des blocs. Concernant ce second problèem, sa $\mathcal{NP}$-complétude est démontrée par un réduction problème de couverture rectangulaire. Formellement, le problème $SBCP$ se définit ainsi :

Notations

> Soit $P = \{(x, y) | M(x, y) \neq 0\}$ l'ensemble des points non nuls de la matrice M et $|P|$ son cardinal.
> Soit $\text{aire}_R(r_k)$, l'aire du k^{me} rectangle (nommé «boîte» et représenté par le quadruplet (a_k, b_k, c_k, d_k)) de l'ensemble R, incluant l'ensemble des points entre les coordonnées (a_k, c_k) et (b_k, d_k) ($a_k \leq b_k$ et $c_k \leq d_k$) des points extrêmes du rectangle. $\text{aire}_R(r_k) = (a_k - b_k + 1) \times (c_k - d_k + 1)$.

La division en blocs d'un espace creux afin de définir un descripteur se modélise par un problème de partition d'une matrice à coefficients dans $\{0, 1\}$ qui représente la signature opposant les valeurs nulles et non-nulles.

Définition 4.1 (SBCP) *Une matrice M de taille $n \times n$, constituée de 0 et de 1 et $K, Q \in \mathbb{N}^*$.*

Question :

Existe-t-il un ensemble R d'au plus K rectangles (boîtes) d'aire supérieure à 1 formant une partition contenant P et dont la somme des aires est inférieure à Q ?

Un ensemble R de boîtes est solution de $SBCP$ si et seulement si :

$$1 \leq |R| \leq K$$
$$\forall r_i, r_j \in R, i, j \in [1, K], i \neq j, r_i \cap r_j = \emptyset$$
$$\forall r_i \in R, \mathit{aire}_R(r_i) > 1$$
$$P \subseteq \bigcup_{i=1,|R|} r_i.$$
$$\mathit{aire}(R) = \sum_{i=1}^{|R|} \mathit{aire}_R(r_i) \leq Q$$

La preuve de $\mathcal{NP}$-complétude peut être trouvée dans [Rem00]. Cette preuve, assez longue, schématiquement réduit ce problème à une variation du problème 3-SAT (définition 4.2) dans le cas où la formule se représente par un graphe planaire. La réduction s'appuie sur une réduction intermédiaire qui est aussi un nouveau problème nommé 3 Positive-Negative Satisfiabilité.

Définition 4.2 (3 Positive et Négative Satisfiabilité)
- *Soit* $V = \{v_1, \cdots, v_n\}$ *un ensemble de variables.*
- *Soit* $\bar{V} = \{\bar{v}_i | \forall i \in \{1, \cdots, n\}\ v_i \in V\}$ *l'ensemble des négations.*
- *Soit* $C = C_1 \cup C_2$ *un ensemble de clauses tel que* $C_1 \neq \emptyset$ *correspond à des clauses de taille* 2 *et* $C_2 \neq \emptyset$ *correspond à des clauses de taille* 3 *tel que toute clause de* C_2 *possède au moins un littéral positif et un littéral négatif, i.e.* $C_1 \subseteq (V \cup \bar{V})^2, C_2 \subseteq (V \cup \bar{V})^3 - (V^3 \cup \bar{V}^3)$,

Question :

Existe-t-il une assignation des variables telle que toutes clauses possèdent un littéral vrai ?

D'un point de vue de l'étude de la calculabilité, certains aspects de ce problème mérite d'être précisé en encart de thématique de ce chapitre. En effet, nous avons prouvé qu'il était $\mathcal{NP}$-complet en le réduisant à Monotone 3-Sat. il s'agit d'un *« assemblage »* de deux problèmes par l'union de deux ensembles de clauses de nature différente. Le problème de la satisfiabilité dans chacun des ensembles peut être résolu par un algorithme en temps polynomial. En effet, si $C_1 = \emptyset$ dans ce cas étant donné que par définition toute clause possède un littéral positif et un littéral négatif, l'attribution de la valeur vraie (ou faux) à toute les variables rend chaque clause de la formule (par exemple $(v_1 \vee v_2 \vee \bar{v}_3)$) nécessairement vrai. Inversement si $C_2 = \emptyset$, chaque clause possède uniquement deux variables. Le problème se ramène alors à $2-SAT$, qui est un problème résolu en temps polynomial. L'assemblage de ces deux problèmes produit un problème $\mathcal{NP}$-complet. S'il fallait le montrer dans le domaine de la calculabilité, ce problème met en évidence que l'assemblage de problèmes simples produit un problème complexe.

La formulation du problème $SBCP$ ne limite donc pas celui-ci à la portée que nous lui donnons de constituer *« bon »* un descripteur mais il définit les contraintes de placement d'une matrice creuse sur une architecture distribuée en respectant une division par bloc. Au delà du partitionnement de matrices creuse, ce type de problème possède des applications dans le traitement d'images pour leur décomposition en régions (par exemple la segmentation d'image [GS90]) et dans la fabrication de circuit VLSI [Lop96]. Il n'existe pas à notre connaissance d'algorithme répondant exactement à ce problème cependant les problème de couverture par polygone et de partitionnement ad-hoc de matrice creuse peuvent être rapprochée de notre problématique. La section 4.4.1 décrit l'état de l'art dans le partitionnement ou la couverture du plan par des polygones. Nous concluerons cette section en montrant l'originalité du problème $SBCP$. Puis nous proposerons une heuristique *à la volée* (on-line) qui partitionne les données au fur et à mesure de la lecture.

$$\left(\begin{array}{cccccccccccc}
0 & 5 & 0 & 0 & 0 & 0 & 2 & 0 & 1 \\
0 & 0 & 8 & 21 & 0 & 45 & 0 & 0 & 2 \\
0 & 7 & 6 & 15 & 1 & 0 & 76 & 0 & 16 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 32 \\
0 & 12 & 0 & 67 & 12 & 8 & 89 & 0 & 43 \\
\\
0 & 1 & 0 & 45 & 0 & 1 & 3 & 0 & 0 \\
12 & 12 & 1 & 0 & 0 & 2 & 0 & 0 & 0 \\
9 & 0 & 0 & 9 & 0 & 57 & 0 & 3 & 0 \\
0 & 8 & 2 & 0 & 0 & 98 & 0 & 0 & 0 \\
18 & 0 & 0 & 8 & 0 & 3 & 0 & 7 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 9 & 0 & 0 & 4
\end{array}\right)$$

La méthode BRD propose un partitionnement par blocs adapté aux matrices creuses. A partir de l'ensemble des valeurs non nulles et d'une grille de processeurs, la méthode MRD coupe la matrice successivement horizontalement et verticalement en divisant chaque bloc rectangulaire ainsi formé en équilibrant le nombre de valeurs non nulles de chaque côté. Cette méthode construit un partitionnement de la matrice creuse en rectangles dont on peut fixer le nombre par la taille de la grille de processeurs. Cependant aucune contrainte n'est imposée sur la surface des rectangles.

FIG. 4.6 – Partitionnement Binary Recursive Decomposition (2×4)

4.4.1 Etat de l'art

Cette section présente des travaux en rapport avec les problèmes de découpage d'un ensemble de points en un ensemble de rectangles.

Dans le cadre des architectures à mémoire distribuée, la partition d'une matrice répond au problème d'équilibre de charge de calcul sur chacun des processeurs. Il s'agit de répartir équitablement les données à traiter afin que les temps d'exécution sur chaque processeur traitant la structure soient approximativement identiques. Dans le cas des matrices creuses, la décomposition est souvent réduite à l'ensemble des valeurs non nulles de la matrice. Ces partitions sont soumises à des contraintes et ont pour objectif de minimiser une fonction de coût comme la répartition du temps de calcul ou la génération d'un minimum de communications induites par le traitement des données. Les méthodes de partitionnement les plus répandues ont comme caractéristique de décomposer les matrices en sous-matrices ou blocs rectangulaires. Ce problème peut se rapporter à celui plus général de décomposition de polygones. Les problèmes de découpage sont $\mathcal{NP}$-complet.

L'algorithme de partionnement BRD et de son extension MRD [CZM94, MU97] (cf. figure 4.6) constitue une classe d'algorithmes possibles. Toutefois ces algorithmes impliquent de posséder une représentation totale des données et ne prend pas en compte les contraintes fixés pour le partitionnement. De plus son objectif n'est de découper la matrice en blocs mais d'équilibrer les valeurs non-nulles.

Les problèmes les plus proche de celui posé par le partitionnement par bloc concernent les problèmes de décomposition de polygones dont l'objectif est de forme une couverture (telle que

chaque point existe dans au moins un des sous-ensemble de points) ou une partition (telle que chaque point existe une et une seule fois dans la réunion des sous-ensembles de points) d'un ensemble de points du plan. Une matrice creuse peut être représentée de manière naturelle sous la forme de *polygones orthogonaux* (rectangles) dont les bords sont des droites horizontales et verticales qui recouvrent les éléments non nuls. Les principaux algorithmes peuvent être classés en fonction de la définition des problèmes auxquels ils se rapportent. Il s'agit pour la plupart de problèmes $\mathcal{NP}$-Complet.

SR25 Soit une matrice de 0 et de 1. Existe-t-il au plus K rectangles formant une couverture des 1 ? Ce problème est $\mathcal{NP}$-complet [GJ79].

SR25P SR25 exprimé en terme de polygones. Soit un polygone orthogonal creux (qui contient des trous). Existe-t-il une couverture d'au plus K rectangles de ce polygone ? Ce problème est $\mathcal{NP}$-complet.

CP Soit un polygone orthogonal creux. Quelle est la couverture en nombre minimal de rectangles de ce polygone ? Ce problème est $\mathcal{NP}$-complet.

PP Soit un polygone orthogonal creux. Quelle est la partition en nombre minimal de rectangles de ce polygone ? Ce problème est Polynomial [LLL+79].

RTILE Soit une matrice de nombres positifs. Quelle est sa partition en au plus p rectangles tels que le poids (somme des valeurs incluses dans un rectangle) de chaque rectangle est minimal ? Ce problème est $\mathcal{NP}$-difficile.

La complexité des problèmes de décomposition de polygones peut être classifiée par les caractéristiques du polygone à couvrir et par la nature du polygone de pavage. La table **??** décrit un synthèse des problèmes rencontrés. Les deux premières catégories présentent des problèmes généraux pour des polygones comportant ou non un trou dans l'ensemble des points à recouvrir. Les deux catégories suivantes introduisent des problèmes plus spécifiques et plus proches du notre. Cependant l'énoncé détaillé de ce problème confirme l'originalité de notre problème qui ne semble pas pouvoir être modélisé par l'un des problèmes énoncés. En effet, une des hypothèses de départ de la construction d'une solution à ces problèmes suppose que les frontières des polygones à décomposer sont connues. La solution donnée est une couverture (ou une partition) telle que tout point du polygone apparaît (une et une seule fois pour une partition) dans un des polygones de la décomposition. De plus, tous les éléments de la décomposition sont inclus dans le polygone originel. Notre problème ne pose pas les mêmes contraintes. La décomposition qui convient au problème $SBCP$ peut comporter des points qui n'existent pas dans le polygone originel sous réserve qu'ils existent dans la matrice. Ces points sont des coordonnées de valeurs nulles. $SBCP$ accepte comme solution des partitions approximant le polygone de départ.

4.4.2 Heuristique de partitionnement à la volée

Le problème $SBCP$, $\mathcal{NP}$-complet, s'étend naturellement à un problème de recherche d'un minimum d'une fonction objective de coût combinant le nombre de données conservées Q et le nombre de partitions K. La caractérisation même d'une bonne fonction de coût minimisant le temps de traitement est difficile à définir. L'objectif est d'établir le facteur prépondérant puis d'estimer la relation entre K et Q. Des expériences préliminaires on montré que le facteur prépondérant était le nombre de 0 en considérant un programme typique de parcours exhaustifs de la structure 4.7. Pour cette mesure, Le *parcours* est un processus qui passe de bloc en bloc. A chaque nouveau bloc, un certain nombre de variables de l'itérateur sont initialisées pour le parcourir. Si on dénote par

polygone à paver	polygone de pavage	Partition Couverture	Complexité	Référence
Polygones pleins				
	convexe	P	polynomial	[CD79],[Cha80]
	en étoile	P	polynomial	[Kei85]
	en étoile	C	$\mathcal{NP}$-difficile	[Agg84]
Polygones creux				
	convexe	P	$\mathcal{NP}$-difficile	
	en étoile	P	$\mathcal{NP}$-difficile	
	convexe	C	$\mathcal{NP}$-difficile	[OS83]
	en étoile	C	$\mathcal{NP}$-difficile	[OS83]
Rectangulaires orthogonal				
	rectangle	C	$\mathcal{NP}$-difficile	[GJ79] (PP)
	rectangle	P	polynomial	[LLL+79]
Spécifique				
matrice à valeurs ≥ 0	rectangle	P	$\mathcal{NP}$-difficile	[KMP98] (RTILE)
l rectangles	k rectangles	P	$\mathcal{NP}$-difficile	[KMP98]
matrice à valeurs dans $\{0,1\}$	rectangle	C	$\mathcal{NP}$-difficile	[GJ79] (SR25)

Cette table décrit les principaux algorithmes de décomposition de polygones en un ensemble de polygones élémentaires tel que l'union des polygones couvre *exactement* le polygone originel. Selon le chevauchement ou non des polygones élémentaires, le problème se présente comme un problème de couverture ou de partition. La valeur à minimiser est le nombre de polygones. Les problèmes spécifiques sont plus précisemment définis dans ce chapitre. Rappelons que :
- un polygone plein correspond à la définition classique d'un polygone.
- un polygone en étoile est constitué de l'intersection de polygones convexes ayant une intersection non vide.
- un polygone creux correspond à un polygone au sein duquel, certaines regions sont manquantes. Ces régions sont correspondent elles-mêmes à des polygones mais en creux.
- des polygones orhogonaux correspondent à des rectangles disposés dans le plan de façon à ce que les arêtes soient parallèles ou orthogonales.

TAB. 4.2 – Problèmes de polygones

```
for(parcours)
    accès SparseArrayM
```

FIG. 4.7 – (prog.) Code générique

C_b le coût de cette initialisation, le *parcours* global a un coût de $K.C_b$ pour un algorithme en $\mathcal{O}(n)$ L'*accès* se contente d'accéder à la valeur référencée par l'itérateur parcourant M. Le coût induit (noté C_c) par cette opération est donc lié aux opérations arithmétiques qui entrent en compte dans le calcul. Le coût global de l'*accès* est alors $Q.C_c$. le temps d'exécution se modélise donc comme une fonction bilinéaire dépendant des variables Q et K :

$$T = K.C_b + Q.C_c$$

Expérimentalement, le coût lié au nombre de valeur à calculer est prépondérant sur le nombre de blocs. L'algorithme de partitionnement mis en œuvre donne donc une priorité à la minimisation de la quantité Q et définit comme secondaire la quantité K. L'algorithme proposé est une heuristique *à la volée* construisant la solution au fur et à mesure qu'il lit les données en entrées. Cette approche apparaît indispensable car la grande taille de certaines matrices creuses interdit de posséder une représentation totale de la matrice sur un processeur. Il faut avoir recours soit à un partitionnement en paralèle soit à un algorithme effectuant la distribution à la volée lors de la lecture des données à partir d'un support externe (fichier). Nous avons choisi la seconde solution. L'algorithme est paramétré par la densité des blocs qui détermine globalement le rapport du le nombre de valeurs non nulles sur le nombre de données à calculer (aire d'un bloc). Dans une première partie, le traitement consiste à agréger les points de la matrice en rectangles en respectant un critère de densité défini localement. Puis les blocs constitués seront de nouveau agréger une méthode similaire dépendant de la densité. Cette second phase permet de déterminer le niveau global du descripteur. Elle agit uniquement sur une représentation sous forme de section de tableaux et non pas sur les données en elle-même. Le principe de constitution des blocs à partir des données est le suivant : En appliquant une lecture de la matrice ligne par ligne, l'algorithme augmente l'aire des blocs déjà constitués pour inclure certains les valeurs de la ligne. Si la densité du nombre de valeurs du bloc est inférieur à une densité de référence. Ce dernier est *fermé* et un nouveau bloc se crée. L'algorithme construit des blocs en les classant en blocs ouverts ou fermés :

- les blocs ouverts sont ceux en cours de construction pour lesquels un point sur une ligne suivante peut-être ajouté.
- les blocs fermés sont ceux ne pouvant grandir de par leur trop faible densité.

Des blocs ouverts sont créés pour les points de la ligne non inclus dans un bloc quelconque. La seule transition possible est donc qu'un bloc ouvert devienne fermé dès qu'il n'est plus possible d'y ajouter une donnée. Le critère de fermeture d'un bloc dépend d'une densité locale au bloc et d'une densité de référence qui peut être arbitrairement fixée. Le choix nécessite une analyse plus détaillée de la topologie des matrices partitionnées. Si l'on observe la figure 4.8 décrivant le profil caractéristique observé expérimentalement de l'évolution du temps avec le nombre de zéro et du nombre de blocs sur des expériences menées pour l'algorithme de parcours typique ; On peut distinguer 3 phases en fonction de l'évolution du nombre de blocs.

- La phase α correspond à un faible nombre de 0. Il correspond généralement à un fort nombre de blocs. Dans ce cas, le coût de gestion des blocs devient important

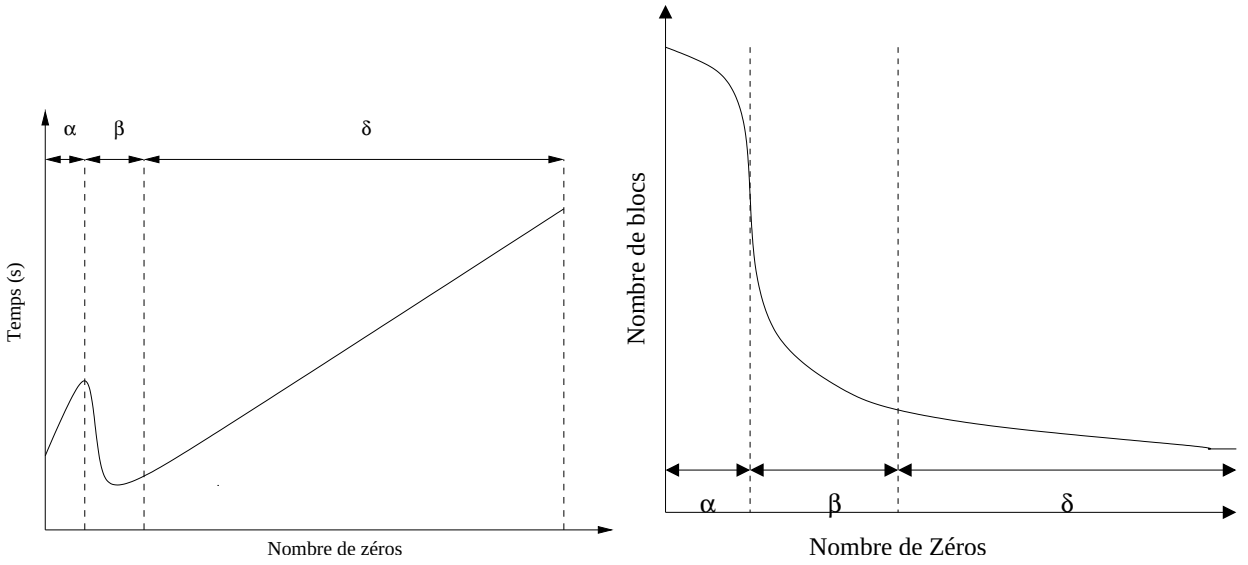


FIG. 4.8 – Courbes schématiques de l'évolution du temps de parcours en fonction du nombre de blocs et du nombre de zéros stockés

- la phase β correspond à nombre de blocs plus réduit sans avoir une augmentation significative du nombre de 0. Cette phase conduit à accélération significative de l'application.
- la phase δ correspond à un faible nombre de blocs. Dans ce cas, le temps d'exécution augmente linéairement en fonction du nombre de 0. L'impact de la gestion des blocs est négligeable face à celui du calcul.

4.5 Etat de l'art

Dans cette section, nous allons comparer des travaux connexes traitant de la programmation parallèle avec la STL, de la gestion des structures irrégulières distribuées et de la compilation des langages data-parallèles pour le traitement des problèmes irréguliers.

4.5.1 STL et parallélisme

La bibliothèque PSTL [JGB97] du projet HPC++ est une extension parallèle directe de la bibliothèque STL. Elle reprend la même architecture logicielle en proposant les mêmes conteneurs et itérateurs que la STL dans des versions parallèles. Les conteneurs de la PSTL peuvent être répartis au travers de *contexte* sur tous les processeurs et leur distribution peut être contrôlée et modifiée à l'exécution. Les itérateurs (ainsi que les algorithmes) existent également en version parallèle. Utilisés en version parallèle, les itérateurs ont une sémantique d'opérations collectives dans laquelle les résultats sur chaque processeur (chaque contexte) sont utilisés pour calculer un résultat global. Dans leur utilisation séquentielle, les calculs sont locaux.

Comparativement à PSTL, nous proposons un conteneur spécialisé pour le calcul distribué et creux auquel est associé un descripteur qui permet d'offrir à l'utilisateur, un espace d'adressage

unique.

4.5.2 CHAOS

Le traitement des structures irrégulières impose de traiter les communications induites par l'accès et le partitionnement irrégulier de ces structures. CHAOS [PHS+95] est un exemple de support exécutif qui permet de transformer des références à un tableau avec des indirections, en communications pour des partitionnements irréguliers. Afin de mieux cerner la difficulté de concevoir un tel schéma inspecteur-exécuteur traitant les indirections [vHKK+93], nous présentons les principales caractéristiques de ce système :

Placement Le placement est représenté par un tableau associé à la structure de donnée à partitionner. Soit un tableau `A`, son placement est stocké dans un autre tableau `Map` tel que si `A(`i`)` est placé sur le processeur p alors `Map(`i`)=`p. Le tableau `Map` qui à la même taille que le tableau `A` est distribué suivant une distribution par bloc puisque généralement il ne peut pas être dupliqué sur tous les processeurs. La distribution régulière est décrite par une fonction (par exemple $where_{Map}(i) = \lfloor i/b \rfloor$ pour un partitionnement en b blocs). En plus du tableau de placement `Map`, un tableau supplémentaire est nécessaire pour mémoriser les emplacements de chaque valeurs dans la mémoire locale.

Référence par indirection Un support exécutif de type inspecteur-exécuteur est nécessaire pour pouvoir définir les communications nécessaires au référence par indirection.

```
forall(i = 0 :n)
  X(e2(i)) = Y(e1(i)) + 1
```

Schéma inspecteur-exécuteur Tous les tableaux dans l'exemple précédent sont distribués. `X` et `Y` sont distribués selon le placement indirect défini respectivement par `MapY` et `MapX`. Les tableaux d'index `e1` et `e2` sont distribués de manière régulière (par exemple par blocs), de telle façon que le placement de chacun de leurs éléments peut être calculé par une fonction. Par conséquent, les autres processeurs ne connaissent pas quelles données sont utilisées pour un calcul particulier puisqu'ils ne possèdent pas ce tableau d'indirections dans son intégralité. Les cinq communications décrites ci-après sont dues à la distribution des tableaux d'indirections et de placement. Le processeur calculant un résultat doit posséder toutes les références aux données qu'il traite mais le partitionnement irrégulier ne permet pas d'appliquer la règle du propriétaire du calcul. Le schéma inspecteur-exécuteur est le suivant :

1. *Localisation de l'information (processeur et adresse mémoire locale) des données référencées.* A partir d'une itération i, le processeur possédant `e1(`i`)` doit déterminer ou est placé `Y(e1(`i`))`. Cette information est obtenue en envoyant une requête au processeur possédant `MapY(e1(`i`))`. Le message en retour est composé de la paire (p, l) qui indique le placement et l'adresse mémoire locale de la valeur.
2. *Collecte des valeurs.* Dès que le placement des données est connu, deux communications (requête puis transmission des valeurs) sont nécessaires pour obtenir les valeurs sur le processeur possédant l'indice `e1(`i`)`.

3. *Calcul et mise à jour.* Le processeur qui effectue le calcul n'est pas toujours le propriétaire de la donnée. Une communication de mise à jour est alors nécessaire.

CHAOS propose donc un schéma en cinq communications dans le cas d'un programme avec des accès par des indirections et une distribution quelconque. PARADEIS se limite à un sous-problème du calcul irrégulier : les problèmes d'algèbre linéaire creuse avec partitionnement définit par l'utilisateur. Dans les problèmes d'algèbre linéaire creuse, l'accès au travers de tableaux d'indirections n'est dû qu'au format de stockage compressé de la matrice. En d'autres termes, le programme effectuerait des accès réguliers si la matrice n'était pas creuse. Le schéma de PARADEIS de part la description des accès conduit à n'effectuer qu'une seule communication.

4.5.3 Structures et compilation des langages data-parallèles pour le traitement des problèmes irréguliers

La bibliothèque CHAOS [WDS+95], [HDS+95], présentée en section 4.5.2, développe un support très général pour des tableaux adressés par indirection où le placement nommé INDIRECT est aussi spécifié par un tableau.

M. Ujaldon et E. Zapata proposent un schéma adapté au traitement des matrices creuses [UZ95], [MU97] pour lequel l'irrégularité des accès découle uniquement du format de stockage. Le module inspecteur-exécuteur supporte l'extension parallèle des structures de stockage creuses telle que CCS (Compressed Column Storage) avec des distributions particulières spécialisées aux programmes d'algèbre linéaire creux. Comme pour PARADEIS, cette approche permet une plus grande efficacité comparée à CHAOS grâce à une adaptation aux structures creuses. Cependant les accès aux structures de données sont toujours exprimées au travers d'indirections dues au format de stockage.

T. Brandes, F. Bréguier & al. [BBCR98] exploitent les types dérivés de HPF pour définir une structure de données creuse. Les motivations de ce travail correspondent aux nôtres : simplifier la programmation parallèle en évitant la présence d'indirections. En fonction de cette structure, les auteurs proposent un support exécutif TRIDENT qui gère les arbres distribués. Bien que notre structure repose aussi sur un arbre, les possibilités et les motivations sont différentes. La structure d'arbre par type dérivé est motivé par son intégration dans HPF. Cette hiérarchie correspond à une organisation d'une structure creuse par colonne puis, pour chaque colonne, par ligne. Dans notre cas, seul le descripteur est hiérarchique et cet aspect est motivé par l'exécution de l'inspecteur en deux étapes : approximation puis raffinement.

4.6 Bilan

PARADEIS étend la bibliothèque STL pour traiter le calcul data-parallèle sur matrices creuses. Les caractéristiques principales de PARADEIS ont été introduites au travers d'exemples simples mais nous l'avons également testé sur des exemples typiques du calcul sur matrices creuses telle que la méthode du Jacobi. La bibliothèque PARADEIS permet une simplicité d'utilisation mais elle peut également être utilisée comme langage de « backend » d'un compilateur automatique. Il s'agit essentiellement d'une étude de faisabilité qui s'inscrit en complémentarité de celle décrite au chapitre précédent. Outre les méthodes nécessaires au fonctionnement en parallèle du descripteur et des accès mémoire, cette bibliothèque intègre un algorithme de partitionnement distribuant les données et construisant les descripteurs. J'ai encadré Didier Remy [Rem00] sur ce sujet. Les concepts

exposés dans ce chapitre se sont concrétisés par la réalisation d'une bibliothèque STL (6000 lignes) en C++ disponibles librement.

Chapitre 5

Langage pour l'annotation des Génomes

Préambule

Ce thème de recherche marque une orientation thématique différente. Ma conversion thématique à la bio-informatique s'est effectuée en 2000. Actuellement, mes travaux de recherche couvrent deux domaines : l'annotation des génomes et la modélisation de l'expression de gènes. Le premier possède les résultats les plus aboutis. Actuellement une thèse est en cours sur une des extensions des travaux sur l'annotation.

Pour une première application à la biologie, le langage TAGCC, j'ai souhaité apporté une contribution qui mette à profit mon expérience de recherche précédente. Elle s'inscrit ainsi dans une motivation et dans une compétence communes aux thèmes précédents : celle de développer un langage dédié à un domaine scientifique.

5.1 Introduction

L'analyse des génomes représente un enjeu scientifique d'importance dont on espère de nombreuses retombées entre autres dans les domaines médicaux, pharmaceutiques, agroalimentaires et biotechnologiques. La systématisation de cette analyse, sa complexité et la taille des données à analyser concourent à donner à l'informatique une place centrale pour un traitement «à grande échelle» des séquences biologiques. Le *séquençage*, c'est-à-dire la lecture des enchaînements de nucléotides constituant les molécules d'ADN des chromosomes et leur numérisation constitue la première étape

de ce traitement. Débuté dans les années 1970, celui-ci est entré dans les années 1990 dans une phase industrielle de numérisation massive ; mettant ainsi à disposition de la communauté scientifique un nombre croissant de séquences. Dans les années 2000, l'«industrialisation» du séquençage met à présent l'éclairage sur l'interprétation de ces données ; ouvrant l'ère de la post-génomique. Cette interprétation implique de repérer les régions *significatives* d'un génome, qui se déterminent par rapport aux rôles biologiques qu'ils assument. L'attribution d'une fonction à une région ou, inversement, la recherche d'une région pouvant posséder une fonction biologique particulière se nomme *l'annotation*. Il s'agit d'interpréter et d'inventorier systématiquement des régions pouvant correspondre à des fonctions biologiques. Dans cet inventaire, la recherche de gènes tient une place centrale.

TAGCC est un langage dédié à l'annotation des génomes qui permet de prototyper rapidement des programmes permettant d'aide à l'annotation. Pour l'exposé du langage TAGCC, nous nous focaliserons plus précisemment sur les méthodes de recherche de gènes.

Deux catégories de méthodes de prédiction de gènes. Actuellement, il existe deux grandes catégories de méthodes pour repérer des gènes : les méthodes par homologie et les méthodes *ab-initio* :

- Les méthodes par homologie tentent d'identifier dans un génome des séquences similaires à celles des gènes déjà découverts. Ces méthodes se fondent premièrement sur la conservation des fonctions des protéines au travers des espèces. Par conséquent, un gène identifié dans le génome d'un organisme servira de références à la recherche d'un gène homologue dans un autre organisme. Deuxièmement elles se fondent sur le fait que des gènes possédant des fonctions similaires possèdent souvent une séquence similaire.
- Les méthodes *ab-initio* déterminent les régions correspondant à des gènes sans utiliser de comparaison directe avec des gènes de références. Elle se fondent en cela sur des critères heuristiques combinant des connaissances de nature biologique et statistique. Ces méthodes reposent pour la plupart sur la reconnaissance d'un agencemement de régions jouant un rôle fonctionnel dans les étapes de transcription et de traduction. (cf. figure 5.1)

Précisons que nous effectuons cette distinction dans un soucis de simplifier la présentation. Les logiciels d'annotations utilisent de manière combinées ces méthodes pour obtenir de meilleurs résultats.

Méthodes par homologie La recherche par des méthodes par homologie constitue une référence en ce domaine. Elle est donc employée de manière systématique pour découvrir des gènes dans un génome [MJ94, LFZRM98, LB98]. L'augmentation du corpus des gènes découverts accroît naturellement les qualités de cette méthode en augmentant le nombre de gènes références pour les comparaisons. Il apparaît difficile de résumer brièvement ces méthodes tant la littérature y est importante (cf. site web [Ana]). Nous nous en tiendrons à l'exposé des principes de reconnaissance par homologie par alignement de séquences. L'alignement de séquence vise à mettre en correspondance les bases deux de séquences différentes. Pour effectuer cette correspondance afin que les deux séquences se ressemblent exactement, certaines bases doivent être soit insérées, soit supprimées, soit substituées (mutées). En considérant un score à ces opérations élémentaires de manipulation des bases, on déterminera une distance caractérisant quantitativement la ressemblance (l'homologie) entre les séquences à aligner. Les modèles de score traduisent la biologie de l'alignement en s'appuyant sur des mesures statistiques pour les établir. Un alignement optimal correspond à un

alignement de score maximal. L'algorithme de recherche de l'alignement optimal est un algorithme de programmation dynamique. Les ouvrages suivants décrivent les méthodes de comparaison par homologie : [Wat89],[DEKM98] pour des ouvrages anglo-saxons et [DF02] pour un ouvrage français.

Bien que constituant une référence en ce domaine, les méthodes par homologie ne relèguent pas pour autant l'autre catégorie à un rang secondaire car les méthodes *ab-initio* complètent les recherches par homologie.

Méthodes *ab-initio*. Elles s'appliquent pour la découverte de gènes sans homologue connu. Par conséquent, les méthodes *ab-initio* se révèlent être particulièrement intéressantes dans une démarche plus prospective de recherche de gènes rares ou de gènes orphelins ne possédant pas nécessairement une fonction connue [BK97, PBG00, SS00, MGRH00]. Elles peuvent aussi agir en complément des recherches par homologie en permettant d'affiner le résultat obtenu. En effet, d'un génome à l'autre, un gène dont la protéine possède une fonction analogue peut cependant différer structurellement. La structure résultante peut alors présenter certaines erreurs qui seront corrigées par des méthodes *ab-initio*.

Le langage TAGCC *propose un environnement de développement de programmes ab-initio.*

Les méthodes *ab-initio* présentent un autre intérêt, plus fondamental, qui réside dans le procédé même de conception : elles impliquent de modéliser la structure d'un gène. Il apparaît en effet évident que seule une analyse partant de la structure puisse être faite sur une représentation symbolique des séquences. Nous entendons par le terme structure, l'agencement de régions caractéristiques qu'un programme de prédiction reconnaît pour trouver un gène. Déterminer un tel programme implique de modéliser cette structure. Actuellement, il n'existe pas de programme qui puisse prédire avec exactitude la localisation des gènes. Toutes précautions gardées, on peut formuler les raisons de cette difficulté ainsi : la notion de gène se détermine essentiellement par une définition fonctionnelle qui fut appelé *le dogme central de la biologie moléculaire* et qui se résume de manière caricaturale par l'équation *un gène = une protéine*. En fait, celle-ci identifie un gène comme étant la séquence capable d'être traduite en une protéine. Cela apporte uniquement un protocole observable de validation d'une séquence par sa traduction en une protéine sans apporter d'éléments définissant sa structure.

Les processus moléculaires associés à l'expression des gènes en protéines reconnaissent des *signaux* qui correspondent à des séquences génomiques jouant un rôle fonctionnel (cf. figure 5.1). L'étude de ces signaux permet de proposer des algorithmes d'annotations basés sur leur reconnaissance dans les séquences numérisées. La structure d'un gène représente dans ce cadre l'agencement de ces signaux. L'état de l'art témoigne à la fois de la variabilité de certains signaux et de la variabilité de la structure en elle-même (le site internet de l'institut Rockfeller détient une bibliographie très complète sur ce sujet [Ana]). Plusieurs signaux et plusieurs structures assument donc la même fonction. Ni l'éventail exhaustif des structures possibles, ni la possibilité de déduire la structure en regard de la fonction ne sont pour le moment établis. La prédiction est alors confrontée au double problème que des séquences, qui ne sont pas des gènes, sont reconnues par les logiciels et que des gènes peuvent ne pas être reconnus par l'excès de contraintes apportées pour les définir. Dans le premier cas le modèle ne discrimine pas assez ; il discrimine trop dans le second.

Aussi, modéliser la structure d'un gène pour obtenir des prédictions plus sures constitue un problème actuel d'importance. La qualité des résultats d'un algorithme sur des séquences tests sert de confirmation au modèle sous-jacent à sa conception.

L'intérêt de découvrir des gènes se trouve renforcé si le programme fournit des éléments permettant d'affiner un modèle caractérisant la structure d'un gène. Remarquons que la recherche par

homologie a essentiellement pour objectif d'identifier un gène par rapport à un autre. Elle ne fournit donc pas d'indication directe sur cette structure. Par conséquent, la découverte d'un gène n'est pas guidée par un modèle. Par comparaison, les programmes ab-initio se fonde sur la définition a priori d'un modèle de gènes pour lesquels les validations confortent ou réfutent le modèle.

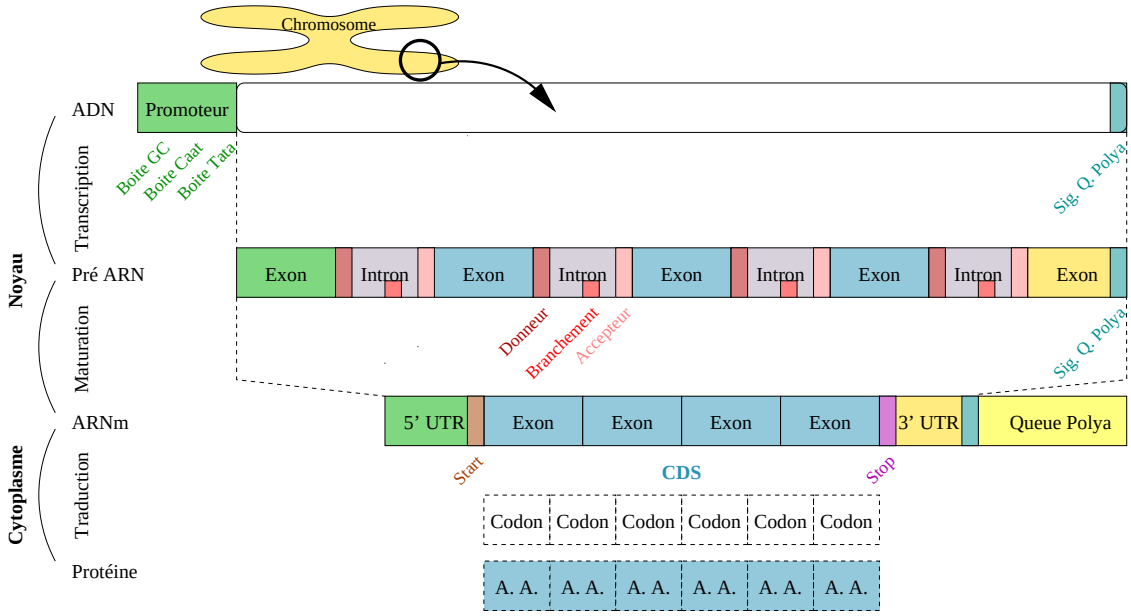
Proposer un langage pour définir un cadre méthodologique à la définition des modèles de structures. Définir un algorithme de prédictions de gènes *ab-initio* devient une entreprise qui nécessite de confronter de nombreux modèles afin de déterminer le plus approprié dans un contexte donné pour la recherche de gènes dans un génome donné. L'absence actuelle de règles universelles conduit à proposer des modèles spécialisés à un génome.

Une étude entreprise en commun avec le Centre National de Séquençage met en lumière que la prédiction de gènes conduit à définir de nombreux programmes, réalisation informatisée de modèles. Si bien qu'en résumé, on pourrait affirmer en écho au dogme central de la biologie moléculaire *un gène = une protéine* que, *un gène = un programme de prédiction.* Les travaux actuels concernant la prédiction mettent plutôt en valeur l'équation *un génome = un programme de prédiction.*

Ceci pose donc le problème de définir des outils théoriques et appliqués pour prototyper rapidement des programmes découlant de modèles. Nous avons choisi de centrer ce développement sur la conception du langage TAGCC spécialisé au domaine de la prédiction. Nous pensons que la conception de langage offre un *lieu* de développement théorique et appliqué qui permet de perfectionner des modèles. Les principes de conceptions de ce langage s'organisent autour de deux motivations très générales : celle de modéliser et celle de représenter. Modéliser vise à décrire par des méthodes informatiques associées au domaine, méthodes elles-mêmes déduites de modèles mathématiques, des instances ou des spécialisations d'un modèle plus générique. En ce sens, la conception du langage vise à définir un cadre générique dans lequel (presque) tous programmes d'annotation puissent être écrits. L'objectif consiste alors à déterminer des *composants* à partir desquels ces méthodes se définissent. Pour mettre en valeur ces composants, nous nous attacherons dans la section 5.2 à décrire les mesures couramment employées pour déterminer si une séquence est codante. Étant admis qu'une mesure ou qu'une classe de mesures peuvent s'employer pour plusieurs méthodes, ce survol permet de dégager des composants communs aux différentes méthodes.

Représenter répond à des considérations plus pragmatiques. Il s'agit de permettre la description concise et *« parlante »* de programmes concernant un domaine. Cet aspect majeur se révèle néanmoins assez subjectif et fait appel à des choix à partir de principes connus des langages de programmations ; choix concernant le style, les relations entre la syntaxe et la sémantique. Nous présenterons dans la section 5.3 les choix effectués pour TAGCC.

Ainsi donc, l'un des éléments essentiels dans la conception d'un langage spécialisé à un domaine est d'identifier un cadre théorique dans lequel les modèles se développent afin d'avoir des programmes adaptés à la modélisation et à la représentation des problèmes qui se posent. Afin de placer TAGCC dans le paysage des programmes et des langages dédiés à l'annotation, nous terminerons ce chapitre par une comparaison de ce langage aux autres langage spécialisés et aux librairies (section 5.6). Dans cette dernière section, nous donnerons des éléments permettant d'apprécier l'efficacité et l'expressivité de TAGCC.



Chez les organismes possédant des cellules à noyaux, les eucaryotes, les processus moléculaires associés à l'expression des gènes en protéine reconnaissent des signaux correspondant à des motifs caractéristiques. Ces signaux fondent en partie les méthodes d'annotations.
L'expression d'un gène inscrit dans l'ADN en protéine nécessite la synthèse préalable d'un ARN par le mécanisme de *transcription*. La protéine responsable de cette synthèse, *l'ARN polymérase* utilise des signaux (ie. des séquences particulières) sur la séquence d'ADN pour déclencher le processus de transcription conduisant à la fabrication d'un pré-ARN messager. Différents signaux ou boîtes composent le *promoteur* qui détermine le début de la transcription. La transcription se déroule dans le noyau de la cellule. Le pré-ARN se compose de régions codantes, les *exons*, et de régions non codantes les *introns*. Durant la phase de *maturation*, les introns sont excisés grâce à un ensemble moléculaire qui reconnaît plusieurs signaux dont ceux bornant les introns : le site donneur (GT) et le site accepteur (AG).
Une fois les introns excisés, la séquence restante forme l'ARN-messager. Une seconde phase nommée la *traduction* débute. L'ARN messager migre dans le *cytoplasme* où il est lu et traduit en une séquence d'acides aminés. Les processus de traduction opèrent par groupe de 3 nucléotides, appelés codons. Des signaux marquent le début START (ATG) et la fin, STOP (TAA,TAG,TGA). A l'exception du codon terminal, STOP, chaque codon correspond à un acide aminé particulier. Le code génétique décrit cette association. Ce code est redondant car plusieurs codons codent le même acide aminé.
il est toujours possible de déterminer un signal consensus à partir de régions dont on sait qu'elles assurent une fonction spécifique,comme le recritement de la protéine de transcription pour le promoteur, mais la réciproque n'est pas nécessairement vérifiée. La présence de signaux ne suffit pas à identifier la fonction d'une région sur l'ADN. L'exemple le plus marquant est certainement celui de la séparation entre introns et exons. Non seulement, un signal GT (resp. AG) peut ne pas correspondre à un site donneur (resp. accepteur), mais il est possible d'avoir des épissages différents à partir de la même séquence d'ARN conduisant à la synthèse de protéines différentes. Ce phénomène se nomme l'*épissage alternatif*. Définition de certains termes :

- nucléotide : une base composant l'ADN, Adénosine, Guanine, Cytosine, Thymine.
- promoteur : séquence initiatrice de la transcription composé de plusieurs signaux nommés *boîte*.
- UTR : (UnTranslated Region) région transcrite mais non traduite
- intron : portion transcrite non-codante contenant 3 signaux le site donneur, le site de branchement, le site accepteur
- exon : portion codante traduite
- CDS : (CoDing Sequence), portion épissée de la séquence codant pour une protéine.
- Queue Polya : Série d'adénosine ajoutée lors de la phase de maturation. Cette séquence joue un rôle dans la stabilité de l'ARN. Un signal nommé sur le schéma Signal de Queue de PolyA (AATAAA) posséderait la double fonction de terminaison de la transcription et de recrutement de la queue polyA.

FIG. 5.1 – Éléments de la structure d'un gène chez les organismes eucaryotes

5.2 Survol des mesures *ab-initio*

Les méthodes d'annotation *ab-initio* se classifient usuellement en deux catégories [Sta84, Fic96] qui se rapportent aux mesures utilisées pour ces méthodes : les mesures statistiques dites *par contenu* (section 5.2.1) qui se fondent sur le contenu en bases A,T,G,C d'une séquence codante typique, et les mesures dites *par signal* (section 5.2.2) qui recherchent des séquences génomiques ayant une fonction connue.

Tout algorithme de reconnaissance de gènes s'appuie sur des mesures calculant le *pouvoir codant* de séquences d'ADN. Il s'agit de fonctions permettant d'évaluer si une partie d'une séquence code pour une protéine. Elle correspond alors à un gène putatif. Les séquences correspondant à des gènes ont la particularité d'être la succession de parties traduites, réellement codantes, les exons, et de parties seulement transcrites, non codantes, les introns (voir schéma 5.1). Chaque séquence codante est séparée par une région intergénique non codante. Le pouvoir codant d'une sous-séquence correspond à une mesure permettant de déterminer si celle-ci fait partie d'un exon. Les méthodes *par contenu* ou *par signal* se distinguent par la manière de calculer ce pouvoir codant.

5.2.1 Mesures *ab-initio* par contenu

L'analyse des séquences génomiques a montré qu'il existait un biais statistique dans la partie exonique des gènes, qui n'existe pas dans les parties intergéniques ou dans les introns. Plus précisément, elles reposent sur l'hypothèse que les séquences non codantes, introns ou régions intergéniques, sont homogènes ; c'est-à-dire que la probabilité d'apparition d'un nucléotide à une position donnée correspond à une équiprobabilité. Par opposition les séquences codantes sont hétérogènes et exhibent un biais sur les positions. C'est pour cette raison qu'ont été introduites les *mesures par contenu*. Ces mesures prennent comme références le contenu typique de séquences exoniques connues ainsi que celui de régions intergéniques ou des introns. Les mesures par contenu se fondent sur l'existence de ce biais pour distinguer une séquence codante d'une autre non-codante. Il existe essentiellement ([Gui99]) deux catégories de mesures par contenu : les mesures qui dépendent d'un modèle d'ADN codant calculées sur la base d'un échantillon et celles se basant uniquement sur l'existence d'un biais.

Les mesures qui dépendent d'un modèle d'ADN codant. Elles se fondent sur un modèle probabiliste décrivant une séquence codante typique. Ces probabilités constituent les paramètres du modèle d'ADN codant. Par conséquent, plus leur évaluation sera précise plus on pourra avoir confiance dans l'opposition prédite entre codant et non-codant. Ce calcul se résume à un calcul de *log-vraisemblance* qui sera plus amplement expliqué à la figure 5.3 où l'on oppose, pour une région, la probabilité d'appartenir à un modèle codant contre la probabilité d'appartenir à un modèle non-codant. La mesure d'appartenance à un modèle codant est fondamentale. Toutes ces mesures exploitent le biais d'utilisation de sous séquences composées de k-nucléotides consécutifs qui composent la séquence codante. Ces mesures se distinguent donc par le choix de ces sous-séquences par rapport à leur taille (k) et à leur fonction. Les principales mesures considèrent la fréquence des codons ($k = 3$), des acides aminés ($k = 3$, avec une transformation des codons en acide aminé), et des héxamères ($k = 6$) qui représentent l'accolement de deux codons. Les programmes employant ces mesures possèdent un noyau relativement identique qui consiste à calculer une log-vraissemblance pour un ensemble de k nucléotides situés dans une fenêtre de taille donnée. Ces méthodes présentent certaines limitations en pratique. L'estimation des probabilités pour ces deux modèles conduit à

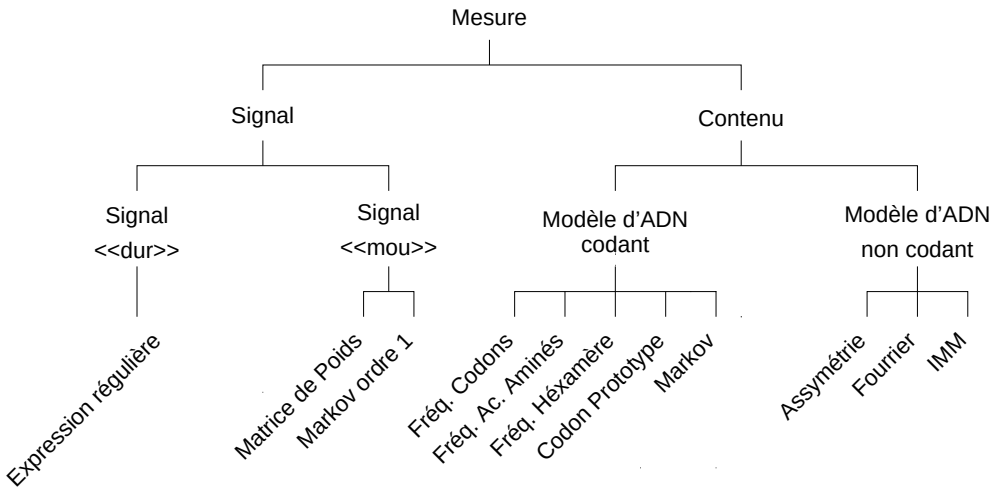


FIG. 5.2 – Paysage des principales mesures des méthodes *ab-initio*

posséder un échantillon d'ADN codant et non codant. Malheureusement, il est souvent impossible de posséder de tels échantillons quand par exemple l'étude d'un génome particulier débute. La détermination des positions des gènes nécessitent une expérimentation lourde et coûteuse qui conduit à posséder un jeu restreint de gènes au début d'un projet d'annotation. Ainsi, il apparaît utile de posséder des mesures de pouvoir codant qui ne dépendent pas de l'existence d'échantillons. Ces méthodes proposent d'établir une discrimination d'un modèle sans échantillon de référence.

Les mesures se basant sur des caractéristiques intrinsèques de l'ADN non-codant. Elles reposent sur la capacité à identifier une région codante en se fondant uniquement sur l'antagonisme intrinsèque du modèle codant et du modèle non codant. Sans échantillon, les scores passés à une fonction déterminant le pouvoir codant n'ont alors pas de signification probabiliste directe, mais leur distribution peut être étudiée empiriquement au sein d'ensembles connus de séquences codantes et/ou non-codante. R. Guigo [Gui99] mentionne trois mesures :

- la mesure d'asymétrie dans la position des bases [Fic82, Sta84] : cette mesure se base sur l'asymétrie dans la distribution des bases au niveau des trois positions d'un triplet d'une séquence ;
- la mesure de Fourier : cette mesure identifie si la décomposition spectrale de Fourier de la séquence correspond à une séquence codante. D'après Guigo et Li ([Gui99, Li97]) les régions d'ADN codant montrent une périodicité caractéristique de 3 car un pic apparaît à la fréquence $f = 1/3$, pic qui n'est pas observées pour les régions non-codantes.
- L'information mutuelle moyenne (IMM). La fonction d'information mutuelle à la distance k est la somme de toutes les corrélations pouvant exister entre deux nucléotides séparés par une distance de k nucléotides. L'information mutuelle moyenne (IMM) se définit alors comme une moyenne des fonctions d'information mutuelle a différentes distances possibles. Cette mesure permet, en observant une périodicité de 3 dans l'ADN codant, d'indiquer si une séquence est codante ou non.

5.2.2 Mesures *ab-initio* par signaux

Les signaux constituent des éléments caractéristiques permettant d'identifier différentes parties de la séquence et de distinguer les régions codantes des régions non-codantes. Leur agencement structure la séquence d'ADN. Celui-ci peut être décrit par une grammaire. Dans ce cadre, la reconnaissance des signaux joue un rôle central en permettant de reconnaître la structure grammaticale de la séquence pour en isoler les régions codantes. Nous pouvons distinguer deux catégories de signaux :

- Les signaux que nous qualifierons de *« durs »*, pour lesquels il existe un nombre fixe et identifié de variations. Par exemple les signaux marquant le début et la fin d'une séquence traduite possèdent un nombre fixé de variations qui sont pour le début ATG et pour la fin TAA,TAG,TGA.
- Les signaux que nous qualifierons de *« mous »* pour lesquels il existe un nombre de variations non fixé, par exemple les sous-séquences du promoteur que l'on appelle aussi *boîtes*. Ces signaux sont donc principalement définis par leur fonction et non pas par leur structure. La classe des signaux associés à une fonction donnée est identifiée par un signal *consensus* qui intuitivement correspond à la séquence où chaque nucléotide apparaît le plus fréquemment. La détection du signal promoteur composés de différentes boîtes représente un exemple caractéristique de l'application de ce type de méthodes. C'est le cas des boîtes des promoteurs en figure 5.1

L'identification des signaux consensus pose plus de problèmes que pour les premiers à cause de leur *« mollesse »*. Il existe pour les analyser différents logiciels qui utilisent des méthodes très similaires fondées sur des matrices de poids et leur extension par des méthodes de chaînes de Markov [DEKM98]. La méthode des matrices de poids s'appuie sur l'étude des fréquence d'apparition des bases en fonction des positions. Plus précisément, à partir d'un échantillon de séquences représentant le même signal, c'est-à-dire ayant la même fonction, et supposé de même longueur n, on relève la probabilité $p(b, i)$ d'apparition de la base b (pour chaque b dans $\{A, T, G, C\}$) en position $i \in [1, n]$. Le signal consensus correspond à une séquence *construite* d'ADN de longueur n où en chaque position i on retient la base b de probabilité $p(b, i)$ la plus élevée (cf. figure 5.3).

5.3 Définition formelle du problème de la prédiction de gènes

La conception de ce langage pose la question d'identifier des structures expressives pour les biologistes en regard des méthodes et des mesures employées dans ce domaine. Pour formaliser cette question, nous avons proposé une définition formelle du problème auquel les méthodes de prédiction de gènes apparaissent proposer une solution ; problème déduit de l'étude résumée dans la section 5.2. Ce problème que nous avons nommé S^2CP (String Segmentation with Constraints Problem) (cf. définition 5.1) détermine plus précisément les composants du langage en les rapportant directement au problème d'annotation des séquences biologiques et non plus aux méthodes. Bien qu'il s'agisse d'une généralisation qui ne prétend pas intégrer exhaustivement toute les méthodes existantes, on peut affirmer que de nombreuses méthodes y trouve un cadre d'expression.

Définition 5.1 (String Segmentation with Constraints Problem)
- *Soit* Σ *et* Γ *deux alphabets finis,*
- *Soit* $\{L_\gamma\}_{\gamma\in\Gamma}$ *une famille de Langages rationnels définis sur* Σ. *(i.e.* $L_\gamma \subseteq \Sigma^*$*).*
- *Soit* $\{p_\gamma\}_{\gamma\in\Gamma}$ *une famille de prédicats indexée par* Γ.

Le problème S^2CP *se définit comme suit : Etant donné un langage rationnel* R *défini sur* Γ^* *et un mot* w *de* Σ^**, existe-t-il un mot* $\gamma_1, \cdots, \gamma_n$ *de* R *et une séquence de sous mots* $w_1, \cdots, w_n$ *tels que :*
- $w = w_1 \cdots w_n$,
- *pour tout* $i = 1 \cdots, p_{\gamma_i}(w_i)$ *est vrai, et*
- $w_i \in L_{\gamma_i}$

De manière informelle ce problème définit l'extraction de la structure d'un gène à partir d'une séquence. Sa définition peut être illustrée par l'étude des étapes algorithmiques de l'identification de la région traduite chez les organismes Eucaryotes. La région traduite débute par un signal ATG. Ensuite, suit une région *exonique* terminant soit par un signal STOP, soit par un signal correspondant à un site donneur d'épissage. Ce dernier identifie une séparation entre deux régions : une région significative pour la production de protéine, l'exon et une région non traduite nommé l'intron. Cette séparation correspond à la suite de nucléotide `GT`. Il est donc possible de décrire ce début de gènes par une expression régulière. `debut = ATG exon GT ...`, avec `exon=(A|T|G|C)+`.

En rapprochant l'équation `début` cet exemple de la définition donnée précédemment, on distinguera :
- $\Sigma = \{A, T, G, C\}$, cet alphabet est celui représentant les nucléotides.
- Pour $\Gamma = \{start, exon, donor\}$, cet alphabet représente différentes régions codantes
- Il est facile alors de définir les langages $L_{start}, L_{exon}, L_{donor}$. Ces langages se définissent naturellement ainsi : $L_{start} = ATG, L_{donor} = GT, L_{exon} = (A|T|G|C)+$
- dans ce cas, w_1 doit correspondre à ATG, w_2 sera une séquence de A, T, G, C et w_3 doit correspondre à GT.

Cependant, le signal GT seul apparaît statistiquement insuffisant pour constituer un critère discriminant un exon. Il est nécessaire d'appliquer d'autres tests pour caractériser la région exonique. Ces tests s'appliquent sur son contenu. Il s'agit par exemple de la fréquence des codons [RG00]. Dans les cas précédemment mentionnés, il s'agit de déterminer une mesure sur une suite de nucléotides ; mesure que l'on compare à un seuil qui permet d'établir quantitativement si la séquence analysée fait ou non partie d'un modèle. En conséquence, des conditions supplémentaires doivent porté dans cet exemple sur w_2. Ces conditions conduisent à calculer une mesure se rapportant aux codons apparaissant dans la séquence que l'on comparera à un seuil. Nous obtenons la définition des prédicats suivants : Soit une fonction α déterminant une pondération de la séquence en fonction de ses codons, on a : $p_{start}(w) = vrai, p_{exon}(w) = \alpha(w) \geq 0, p_{donor}(w) = vrai$

En résumé, *l'objectif est de trouver un découpage de la séquence en régions qui convienne aux différents tests possibles pour caractériser ces régions.* Ainsi, le langage TAGCC a pour objectif de fournir un cadre de description et de résolution aux problèmes formulés comme un problème S^2CP. La définition de ce dernier détermine les composants de TAGCC. Ils concernent :
- la description de structures grammaticales,
- la définition de calculs sur les mots,
- la possibilité de définir des conditions portant sur les calculs précédemment définies

D'un point de vue informatique, ce problème met en valeur la nécessité d'intégrer aux mécanismes d'analyse grammaticale des mécanismes de résolution de contraintes. Techniquement, les résultat de la compilation d'un programme TAGCC est un transducteur non déterministe intégrant des contraintes. Les programmes que nous avons eu à développer en collaboration avec les biologistes ont pu être écrit en se limitant à ce cadre. Comme nous le verrons plus loin nous utilisons les

transducteurs pour proposer plusieurs solutions à un même problème. Ce point est important pour la prédiction de gènes car de nombreux phénomènes moléculaires sont inconnus et peuvent conduire à avoir plusieurs annotations pour une même séquence. Choisir l'une des annotations dépend d'un contexte encore très mal connu (épissage alternatif, structure de l'ADN, variation entre espèce, etc.). L'un des principaux apport de TAGCC est de marier la multiplicité des solutions à une efficacité grace à l'utilisation des transducteurs. En se rapportant aux composants de TAGCC, les structures grammaticales correspondent à aux expressions régulières, le calcul sur les mots aux transducteurs, enfin les conditions à un mécanisme original que nous exposerons en

La section 5.4 décrit les composants du langage TAGCC. Chacun des composants sera illustré par des exemples réalistes dans le domaine de l'annotation (cf. figure 5.3). Au cours de cette description, nous reviendrons plus en détail sur l'intégration de ces mécanismes.

5.4 Description de TAGCC

TAGCC est un langage équationnel dont nous allons introduire les trois composants issus du problème S^2CP. Nous les illustrerons sur des problèmes pratiques rencontrés dans le domaine de la prédiction de gènes.

5.4.1 Equation et Expression régulière

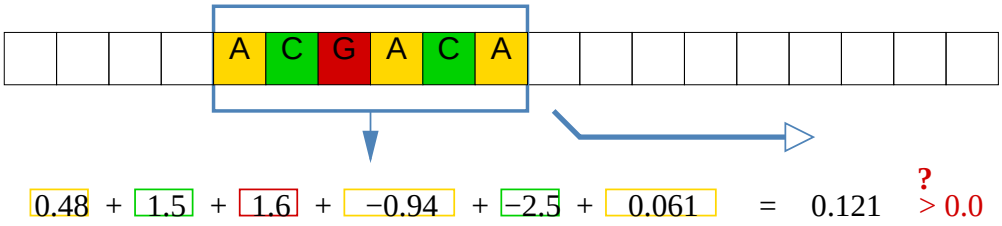
Les équations composant les programmes TAGCC sont définies par des *expressions régulières* suivant la syntaxe suivante :

$$< signal > = < définition > \ ;$$

Un programme TAGCC est une suite d'équations. La transparence référentielle des langages équationnels permet d'avoir des définitions formelles des différentes équations où un nom peut être remplacé complètement par sa définition.

L'emploi d'un langage rationnel répond au besoin d'identifier des séquences comportant des signaux. Il s'agit d'un langage destinée à la programmation de méthodes algorithmiques *ab-initio* utilisées pour la recherche de gènes. Par exemple, le programme 5.4 décrit le programme TAGCC qui recherche les CDS (CoDing Sequences), à savoir une séquence pouvant potentiellement contenir un gène. Nous ne prétendons pas modéliser les phénomènes biologiques par ce langage. Ces équations servent à la définition des langages L_γ du problème. Cependant, la correspondance entre les équations et ces langages n'est pas aussi directe. Elle sera exactement précisée dans la partie 5.4.4.

$$\beta = \begin{array}{c} A \\ C \\ G \\ T \end{array} \begin{pmatrix} 0.48 & -2.5 & -\infty & -0.94 & 1.7 & 0.061 \\ -0.52 & 1.5 & -2.5 & -\infty & -2.5 & 0.28 \\ -0.94 & -\infty & 1.6 & -\infty & 1.7 & -0.52 \\ 0.48 & 1.8 & -0.52 & 1.9 & -2.5 & 0.061 \end{pmatrix}$$



0.48 + 1.5 + 1.6 + −0.94 + −2.5 + 0.061 = 0.121 ? > 0.0

La reconnaissance d'un signal consensus par une méthode du ratio de log-vraissemblance consiste à discriminer un signal en examinant le rapport entre la mesure statistique d'appartenir à un modèle et celle de son contre modèle. Soit une séquence S, on détermine $P(S|M)$ la probabilité qu'une séquence observée correspond au modèle M (à un promoteur pour notre exemple). $P(S|\bar{M})$ correspond à la probabilité de ne pas appartenir au modèle M.
On note par $p_{(b,i)}$ la probabilité d'occurrence d'un nucléotide b en position i dans le signal et par p_b la probabilité d'apparition d'un nucléotide b dans le génome de l'organisme étudié. Celui-ci peut dévier significativement d'une mesure d'équiprobalité (0.25). L'identification du signal par la méthode du ratio de log-vraissemblance e correspond à la fraction : ($|S|$ dénote la taille de S)

$$r = \log \frac{P(S|M)}{P(S|\bar{M})} = \sum_{i=1}^{|S|} \log \frac{p_{(b,i)}}{p_b}$$

En posant $\beta_{b,i} = log\frac{p_{(b,i)}}{p_b}$, on a $r = \sum_{i=1}^{|S|} \beta_{i,b}$. Si r est positif le signal fait partie du modèle. β correspond à une matrice de poids. Pour cet exemple, nous avons pris la définition du promoteur de l'organisme *E. Coli* [SH89]. Le programme TAGCC décrivant cette méthode est le 5.7.

FIG. 5.3 – Méthodes de calculs par matrice de poids

Descriptif des Opérateurs

Cet encart décrit succintement les principaux opérateurs de TAGCC. Ils correspondent à ceux que nous avons utilisés pour décrire des signaux. Mais cette liste ne prétend pas être exhaustive. L'alphabet correspond aux lettres symbolisant les nucléotides, à savoir `A,T,G,C`.

Concaténation ($< exp > < exp >$) **:** décrit la succession d'expressions régulières. Il permet de définir des signaux. Par exemple, la définition du codon start sera donné par : `Start = A T G ;`

Alternative ($< exp > | < exp >$ **ou** `[` $< exp > \cdots < exp >$ `]`) **:** détermine un choix. Il existe deux notations dont les résultats sont identiques. Des signaux possédant plusieurs définitions peuvent ainsi être déclarés. Par exemple le symbole `N` représentant tous les nucléotides sera définis ainsi : `N = [A T C G] ;` ou `N = A|T|C|G ;`

Fermeture de Kleene ($< exp >$ `*` et $< exp >$ `+`) **:** déterminant les répétitions infinies. Par exemple, une succession de longueur inconnue de nucléotides sera décrite ainsi : `exon=N+ ;`

Répétition Finie ($< exp >$ `*` $< int >$ et $< exp >$ `*{` $< int >$ `:` $< int >$ `}`) Deux répétitions d'un motif sont possibles. La première indique n répétitions d'une expression, n étant fixé ; la seconde n étant variable entre deux bornes.

Complément sur l'alphabet (`- -` $< exp >$) **:** Détermine le complément d'un ensemble de lettres sur un sous ensemble de l'alphabet. Par exemple, les purines correspondent au complément des pyrimidines sur N. `Pu = [A G]; Py = --Pu;`

Miroir ($< exp >$ `<-`) **:** permet d'inverser le sens de lecture d'un signal. Par exemple : `Signal2 = (A T G)<- ;` $\equiv$ `Signal2 = G T A ;`

Substitution ($< exp > \backslash\{(exp_1^1, exp_2^1) \cdots (exp_1^n, exp_2^n)\}$) **:** Substitue les éléments d'indice 1 présent dans un signal par les élements d'indice 2 ; par exemple : `(A T G) \{(A,T) (T,A) (C,G) (G,C)}` $\equiv$ `(T A C)` Notons que la conjonction du miroir et de la substitution que nous avons définies précédemment permet d'obtenir la séquence complémentaire. En appliquant cette combinaison, une méthode d'analyse de signaux définie par une équation peut aussi s'appliquer sur la séquence complémentaire car les signaux de l'équation sont transformés en leur complément.

La reconnaissance de séquences à partir d'expressions régulières structurées en equations offre une clarté dans la définition des signaux nécessaires à l'identification de séquences. L'utilisation d'expressions régulières pour décrire la structure syntaxique de séquences est une méthode couramment employée dans ce domaine. La syntaxe de TAGCC s'inscrit donc dans la pratique d'outils informatiques pour ce domaine. Le style déclaratif permet d'utiliser les fichiers de données caractérisant les motifs d'un site fonctionnel comme un programme. En effet en TAGCC, une liste de signaux constitue un programme. Toutefois, la recherche de séquences faites à partir d'une reconnaissance d'expressions régulières apparaît insuffisante. Notamment, il est impossible d'exprimer des calculs nécessaires à la prédiction tel que ceux prenant comme critère de décision la fréquence de signaux particuliers. Prenant pour base la syntaxe d'expressions régulières, nous avons étendu ce langage pour permettre de définir des calculs et des conditions sur les séquences.

5.4.2 Calculer sur des séquences

La déclaration équationnelle procure une grande clarté dans la définition des signaux. Son extension pour intégrer d'abord des calculs, puis des contraintes répond aux insuffisances d'expressions des langages rationnels pour ce domaine. L'approche que nous allons décrire pour intégrer ces calculs au langage s'inscrit dans la volonté de concilier dans les outils, expressivité, efficacité et formalisation. La conception du langage est donc guidée autant par la recherche d'expressivité que la recherche d'efficacité. Dans cette perspective, l'utilisation des expressions régulières répond à ces contraintes car sa compilation produit un automate. La reconnaissance à partir d'un automate est

```
N =[A T G C];
Start = A T G;
Stop  = (T A A) | (T A G) | (T G A);
cds53 = N*  Start N+ Stop  N*;
cds35 = (cds53 <-) \{(A,T) (T,A) (G,C) (C,G)};
cds = cds53 | cds35;
cds;
```

Dans ce programme deux équations caractérisent une CDS (CodDing Sequence). Le premier (`cds53`) s'applique à la reconnaissance dans le sens 5'3' alors que le second (`cds35`) s'applique à la reconnaissance dans le sens contraire 3'5'. En l'absence d'informations supplémentaire, une séquence numérisée peut représenter soit le brin 5'3' ou celui 3'5'. Dans ce cadre, la séquence principale `cds` tient compte des deux possibilités.

FIG. 5.4 – (prog.) Reconnaissance des CDSs en TAGCC

de complexité $O(n)$ où n est la longueur de la séquence ; ce qui est le meilleur ordre de grandeur en temps que l'on peut obtenir. L'intégration des calculs doit permettre de conserver l'efficacité de ce traitement. Elle s'oriente donc vers une extension des automates, les automates pondérés (Weigthed Finite State Automata) [RS97]. Comparativement à d'autres travaux portant sur la définition de langages pour l'analyse de séquences biologiques [Sea94] l'emploi d'automates permet d'avoir des algorithmes très performants. En effet, la compilation de ce langage produira un automate pondéré non déterministe. Or, l'utilisation d'un automate pondéré permet d'obtenir un algorithme le plus performant en complexité pour calculer des fonctions sur les mots. En conséquence, les traitements se rapportant strictement à des calculs de fonctions sur les mots sont très performants [MPR00, MS96, Moh96, Moh97].

Ceci ne veut pas dire que des programmes de découverte de gènes programmés en TAGCC possèdent la garantie d'être les plus performants. Mais cette garantie se porte sur les sous traitements de calculs de poids sur les mots. Pour ces traitements couramment utilisées au sein d'algorithmes de ce domaine, la performance est optimale en terme de complexité temporelle car elle est linéaire en fonction du mot en entrée. La section 5.6 développe plus en détail cette comparaison. En résumé, architecturé autour des automates pondérés TAGCC vise à fournir un traitement efficace à l'implantation de fonctions et plus généralement de relations sur les mots.

Le théorème de Kleene [RS97] [Aut94] établit l'équivalence entre les langages rationnels descriptibles par des expressions régulières et les langages reconnaissables par un automate à états finis. La compilation traduit une expression en automate. Les relations rationnelles sont équivalentes aux automates pondérés. La définition 5.2 définit inductivement les relations rationnelles.

Définition 5.2 (Relation Rationnelle ou Transduction Rationnelle)
Soit Σ un alphabet, soit $(\mathbb{D}, \otimes, \bar{1})$ un monoïde, soit $R \subseteq \Sigma^* \times \mathbb{D}$ et $R' \subseteq \Sigma^* \times \mathbb{D}$ deux relations, on dénote par :
- $R.R' = \{(r, r') | \exists (u, v) \in R, (x, y) \in R', r = u.x, r' = v \otimes y\}$
- $R|R' = R \cup R'$
- $R^n = \{(r_1.r_2 \cdots r_n, \bigotimes_{i=1}^{n} r'_i) | (r_i, r'_i) \in R, \forall i \leq n\}$
- $R^+ = \bigcup_{n>0} R^n$
- $R^* = (\epsilon, \bar{1}) \cup R^+$

Nom	Domaine de définition	Opération	Élément neutre
`max01`	$[0..1]$	maximum (max)	0
`nat`	$\mathbb{N}$	addition (+)	0
`real`	$\mathbb{R}$	addition (+)	0
`rmin`	$\mathbb{R}^+$	minimum (min)	$+\infty$
`seq`	$\{A, T, C, G\}$	concaténation (.)	mot vide

TAB. 5.1 – Liste des opérations implantés dans TAGCC

Cette définition exhibe une forme particulière du calcul qui se couple à la reconnaissance faite par l'automate. Une relation rationnelle détermine donc un calcul implicite qui s'opère au fur et à mesure que l'algorithme de reconnaissance progresse dans l'automate. Avec l'emploi des relations rationnelles, *le calcul ne se programme pas, il se déclare.* Les variables constituant le support du calcul se déclarent non plus en fonction d'un domaine de valeur mais en fonction d'un couple domaine opération. Il correspond à une structure algébrique de monoïde $< \mathbb{D}, \otimes, \bar{1} >$ où $\bar{1}$ est l'élément neutre du monoïde. TAGCC comporte un ensemble de monoïdes qui permettent d'effectuer les calculs les plus courants pour la prédiction de gène (table 5.1). Le programme 5.5 de calcul de la longueur d'une séquence et de compte du nombre de `G` et de `C` illustre un calcul utilisant ces monoïdes. A chaque occurrence d'une expression régulière, il est possible d'associer la valeur particulière d'une variable. Lors de la reconnaissance, le traitement appliquera l'opération $\otimes$ à chacune des occurrences des valeurs reconnues. Ce calcul est fidèle à la définition des relations rationnelles en posant par convention que toute expression ne possédant pas de valeur assignée aux variables ($l = \cdots$) correspond à une assignation à l'élément neutre ($l = \bar{1}$). Par exemple, sur la séquence `ATGCTGAGTA` le résultat de l'exécution du programme 5.5 donnera les valeurs suivantes `len=10, gc=4`. Une définition formelle peut être trouvée dans [Del01].

L'élargissement des langages rationnels aux relations rationnelles dote donc le langage de capacités de calculs nécessaires aux méthodes d'annotation. Pour le problème S^2CP, ce calcul constitue une des composantes de la formulation des conditions caractérisant les prédicats en permettant d'opérer des calculs qui seront confrontés par exemple à des seuils. Cet élargissement n'est cependant pas sans conséquence :

Premièrement, l'emploi de relations pour prédire la structure d'un gène implique l'existence de plusieurs réponses pour une séquence à analyser. Caractériser ainsi un algorithme de recherche de gènes par une relation formalise l'imprécision de l'analyse mentionnée en introduction de ce chapitre ; imprécision qui peut être imputée à l'incomplétude des modèles utilisés pour l'annotation. Définir un programme caractérisant une relation peut cependant s'avérer utile même si le modèle est exact. Elle peut exprimer l'existence de plusieurs gènes entrelacés dans une même séquence. Ce phénomène correspond au phénomène biologique de l'épissage alternatif où selon des conditions spécifiques, différents gènes sont traduits à partir d'une même séquence [Lew00]. Aussi, formaliser l'algorithme de prédiction d'un gène sous la forme d'une relation apparaît adéquat car il recouvre une réalité biologique. Cette réalité se définira naturellement mathématiquement par une relation qui à une même séquence d'entrée identifie plusieurs séquences de sorties possibles représentant différentes combinaisons d'exons. Chaque combinaison correspond à un gène particulier. Précisons que cette relation peut aussi s'interpréter comme une absence d'information permettant de distinguer les solutions.

Deuxièmement certains théorèmes fondamentaux sur les automates classiques ne sont plus véri-

```
1   nat gc,len;
    N = [A:{len=1} T:{len=1} C:{gc=1,len=1} G:{gc=1,len=1}];
    Sequence=N*;
    Sequence;
```

```
1   nat gc,len;
    N=[A T C:{gc=1} G:{gc=1}]:{len=1};
    Sequence=N*;
    Sequence;
```

La variable `len` est associée à chaque base tandis que `gc` ne sera augmentée de 1 que lorsque G ou C seront reconnus dans la séquence. Dans la seconde version du programme `len` a été associée aux quatre bases mais cette fois-ci en la *« factorisant »* ; c'est-à-dire en appliquant ce calcul globalement à l'ensemble des bases.

FIG. 5.5 – (prog.) Mesure de la longueur et du nombre de G et de C

fiés sur les automates pondérés. En particulier, l'un des théorèmes est la possibilité de déterminiser l'automate [ASU86]. Cette possibilité n'est plus toujours vérifiée dans le cadre des automates pondérés. Il existe donc des relations rationnelles dont l'automate sous-jacent ne peut être déterminisé [RS97]. Un exemple de programme TAGCC correspondant à ce type de relation est le suivant : `relation = A* (A :l=1)*`. Si l'on considère la séquence `AAAAA`, il existe 5 solutions possibles pour `l` ($l \in 0, 1, 2, 3, 4, 5$). Plus généralement le nombre de solutions de cette équation dépend de la longueur de la séquence en entrée. On ne peut donc borner le nombre de solutions a priori. Ce type de relations où la cardinalité de l'image ne peut être bornée par une constante ne peut correspondre à un automate déterministe pondéré. Nous en avons donné une démonstration dans [Del01]. Cet exemple illustre le problème technique auquel nous sommes confronté : *devoir gérer efficacement la reconnaissance par des automates pondérés non déterministes.* Nous avons vu que de tels automates correspondent à des besoins effectifs pour la recherche de séquences biologiques.

Pour l'efficacité de ce traitement nous avons choisi une méthode permettant de *« déterminiser le plus possible »* un automate pondéré en conservant le minimum de points d'ambiguité. Précisons que la déterminisation ou d'identifier des classes d'automates dans lesquels on peut appliquer un algorithme de déterminisation des automates pondérés sont des problèmes importants de cette thématique. (cf. section 5.6) ou à. Par conséquent, la reconnaissance/calcul par des automates pondérés non déterministes apparaît un champ d'études nouveau dont l'intérêt se justifie ici dans le contexte d'un langage dédié à la prédiction de gènes, mais dont les résultats peuvent présenter un intérêt général en informatique. La solution que nous proposons va nous permettre de répondre à la fois au problème de reconnaissance par un automate non déterministe et à celle d'intégrer des contraintes dans l'analyse. Elle se fonde sur une notion originale d'emploi de *marques* (ou de balises). Les marques possède un double rôle : le premier concerne la définition d'une méthode pour traiter les automates pondérés non-déterministes ; le second est de permettre l'intégration de contraintes dans l'analyse faites par un automate. Dans la section suivante nous décrirons la manière dont les marques sont utilisées dans ce double emploi.

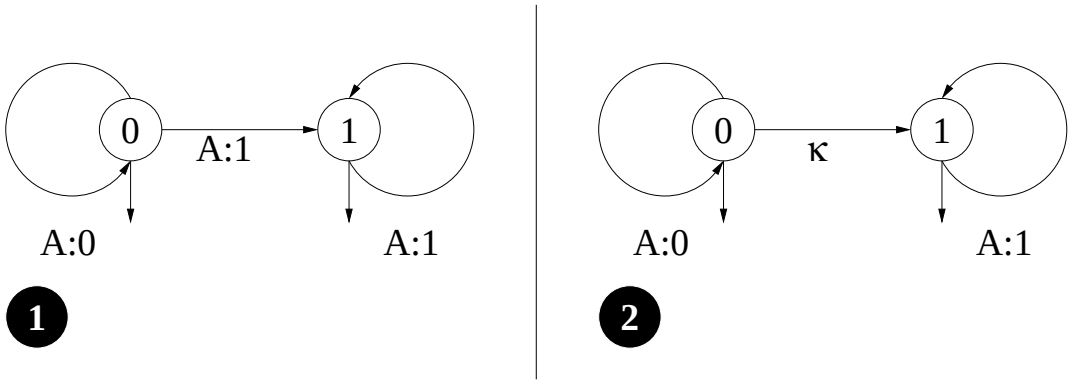


FIG. 5.6 – Exemple d'automates pondérés avec ou sans marque

5.4.3 Marquage et reconnaissance non déterministes

Considérons le programme TAGCC suivant qui ne peut être analysé par un automate pondéré déterministe. `equ=A*A :{w=1}*`. On remarquera que pour une séquence faisant partie du langage A^*, le nombre de solutions correspond à la longueur de la séquence. Nous avons prouvé dans [Del01] que cette catégorie de relations que nous avons nommée *« relations infiniment valuées »* ne peuvent correspondre à un automate pondéré déterministe car le nombre de sorties ne peut être borné par une constante, ce qui est une condition nécessaire pour déterminiser un automate pondéré. L'automate correspondant à cette relation rationnelle correspond au premier automate de la figure 5.6.

Une marque est un symbole supplémentaire qui, introduit dans l'automate pondéré sert à déterminiser artificiellement celui-ci. Pour donner une approche intuitive d'une marque, celle-ci peut être considérée comme un signe de ponctuation comme pour ôter une ambiguïté dans l'interprétation d'une phrase. La syntaxe d'une marque est `/{`$< nom >$`}`.

Dans l'exemple, l'introduction d'une marque sert à séparer la répétition de `A` pondérés de la répétition de `A` non pondérés. On a : `equ = A* /{`κ`} A :{w=1}*`. L'automate correspond au second automate de la figure 5.6. Celui-ci est déterministe.

En résumé, une marque est insérée en des points présentant une ambiguïté, ce qui conduit à déterminiser l'automate pondéré. Bien que l'on puisse avoir plusieurs noms pour identifier les marques, il faut noter que celles-ci correspondent toutes à un unique symbole (κ) représentant le marquage de manière générique. Les différents noms que nous attribuerons aux marques ne sont que du *« sucre syntaxique »* facilitant la programmation. C'est donc en fonction d'un unique symbole ajouté à l'alphabet que le marquage peut être défini. Ceci revient à insérer des $\epsilon-$transitions[1] dans l'automate pondéré. Nous avons cependant voulu les distinguer des ϵ-transition avec la lettre κ à cause du traitement spécifique lié au marquage que nous détaillons à présent.

Bien que l'introduction de marques puisse produire un automate marqué pondéré déterministe, il n'en demeure pas moins que la reconnaissance doit conduire au calcul d'une relation où le nombre de solutions par séquence correspond à la longueur de la séquence lue. Pour ce faire, le lecteur (l'analyseur de séquence) sera non-déterministe. La solution repose sur une double lecture des états

[1] ϵ désignant le mot vide

possédant une marque. A chaque état possédant comme transition une marque, le lecteur opère une double reconnaissance : celle considérant la transition étiquetée par la marque et celle considérant la transition correspondant à la lettre courante. Cette double lecture peut être représentée par un arbre déterminant les différents choix opérés. Chaque arc de cet arbre est étiqueté par l'étiquette de la transition franchie dans l'automate. La concaténation des étiquettes des différents chemins correspond donc à la séquence lue augmentée des marques introduites lors de la lecture.

Ce chemin décrit une *interprétation* de la séquence qui formellement correspond à un ensemble de couples (marque, position). A chaque interprétation correspond une pondération unique ce qui permet de déterminiser l'automate pondéré.

En d'autres termes, le processus de marquage transforme une *relation* en une *fonction* dans un langage augmenté d'un symbole caractérisant de manière générique la notion de marque. Formellement, il vise à définir une relation rationnelle de $R : \Sigma^* \to \mathbb{D}$ comme la composition d'une relation T et d'une fonction F qui peut être décrite par le diagramme suivant :

$$\begin{array}{ccc} \Sigma^* & \xrightarrow{R} & \mathbb{D} \\ {\scriptstyle T}\downarrow & & \uparrow{\scriptstyle F} \\ (\Sigma \cup \{\kappa\})^* & \xrightarrow[\mathsf{Id}]{} & (\Sigma \cup \{\kappa\})^* \end{array}$$

En réalisant cette décomposition, la déterminisation de l'automate pondéré devient possible car une fonction rationnelle se représentera par un automate pondéré déterministe. Le traitement relatif à la relation, c'est-à-dire la production de plusieurs résultats à partir d'une seule séquence est fait par le lecteur. En fait l'utilisateur définira en partie la relation T en introduisant les marques désignant les points de choix possibles.

Ceci a une conséquence importante ; la déterminisation permet de produire des exécutables très performants car, hormis les transitions correspondant aux marques, la reconnaissance se comporte de manière similaire à un automate déterministe classique. Cette question sera plus amplement discutée dans la section 5.6.

En considérant le problème S^2CP, les marques définissent l'alphabet Γ. Elles permettent de mettre en valeur la structure trouvée à partir des équations du langage. Les composantes du problème S^2CP sont donc presque toutes caractérisées, à l'exception des prédicats. Nous allons aborder la manière dont les prédicats se définissent dans le langage TAGCC.

5.4.4 Marque et conditions

Le rôle des conditions est de valider la reconnaissance en appliquant des conditions sur les quantités calculées lors de l'analyse. Les transitions marquées permettent d'intégrer de telles contraintes. En effet, la possibilité de franchir une transition marquée donne lieu à une lecture non conventionnelle de la séquence en fonction de l'automate. La suite de la séquence peut alors être lue soit à partir de l'état courant, soit à partir de l'état cible de la transition marquée. Le passage d'une transition marquée correspond à une alternative dans les solutions possibles. Pour tenir compte des conditions associées aux marques, une transition marquée devient franchissable si la condition qui lui est associée est satisfaite [RS97] .

L'intégration des contraintes augmente significativement l'expressivité du langage dans le sens où des algorithmes fréquemment utilisés dans le domaine de la prédiction de gènes sont fondés

```
real p;
profile  =
 [A:{p=0.48}  C:{p=-0.52} G:{p=0.94}  T:{p=0.48} ] //  1
 [A:{p=-2.5}  C:{p= 1.5}              T:{p=1.8}  ] //  2
 [            C:{p=-2.5}  G:{p=1.6}   T:{p=-0.52}] //  3
 [A:{p=-0.94}                         T:{p=1.9}  ] //  4
 [A:{p=1.7}   C:{p=-2.5}  G:{p=-2.5}  T:{p=-1.5} ] //  5
 [A:{p=0.061} C:{p=0.28}  G:{p=-0.52} T:{p=0.061}];//  6
N=[A T G C];
scanning =  N* profile {p > 0.0 -> PROMOTER}  N*;
```

Ce programme est celui implantant la méthode décrite à la figure 5.3. On remarque dans la matrice la présence de $-\infty$ qui, clairement, exprime l'absence de ce nucléotide. Dans le programme TAGCC, cet élément n'est pas intégré car l'utilisation d'un automate pondéré permet de supprimer les nucléotides n'apparaissant pas. Le programme TAGCC décrit le signal consensus avec une syntaxe simple. (On peut assimiler ce programme à la description de la matrice transposée de β de la figure 5.3). Les commentaires sont inscrits après //. Les numéros de 1 à 6 après les signes de commentaires indiquent la position de la lettre analysée au sein du signal. L'équation `scanning` correspond à l'équation principale et marque la présence d'un promoteur lors de l'analyse sur une séquence.

FIG. 5.7 – (prog.) Programme TAGCC de détection d'une séquence consensus

sur la reconnaissance de signaux auxquelles s'ajoute la caractérisation d'un seuil s'appliquant sur une pondération calculée à partir des signaux. Il s'agit en particulier de l'identification de signaux consensus par des chaînes des matrices de poids (cf. figure 5.3).

Nous allons illustrer l'emploi des conditions dans ce cadre par un exemple classique d'identification : la recherche d'une séquence consensus par une méthode fondée sur une matrice de poids ; méthodes aussi nommées méthodes de Markov d'ordre 0.

Identifier un signal qui ne suit pas une définition stricte rend la recherche plus difficile car des variations de ce signal apparaissent. Rappelons que la notion de signal n'est pas associée à une syntaxe mais à la fonction d'une région de la séquence d'ADN. Aussi, à une fonction unique différentes syntaxes peuvent correspondre. Pour déterminer de tels signaux, on fait appel au calcul de log-vraissemblance permettant la discrimination de signaux. Cette méthode utilise des *matrices de poids* pour implanter l'algorithme de discrimination. Le programme 5.7 décrit cette méthode. L'extension de cette méthode consiste à utiliser les chaînes de Markov pour discriminer un signal. Il s'agit d'une extension classique qui doit être implantable en TAGCC. Dans ce cadre, les méthodes utilisant les matrice de poids s'assimilent à une chaîne de Markov d'ordre 0 (sans mémoire). En utilisant les chaînes de Markov d'ordre supérieur $(n, n > 0)$, on étend la probabilité d'avoir une lettre à une position donnée à une probabilité dépendant aussi des n lettres la précédant. Nous renvoyons le lecteur à l'ouvrage de R. Durbin et al. (chapitre 3) [DEKM98] pour plus de précisions sur la manière dont cette extension s'effectue. La description d'un programme TAGCC pour la reconnaissance de signaux consensus en utilisant une matrice de poids (chaîne de Markov d'ordre 0) se généralise à des chaînes de Markov d'ordre supérieur. *Toutes méthodes d'identification de signaux consensus par discrimination utilisant les chaînes de Markov peut être réalisées par un programme* TAGCC. Celui-ci consiste à décrire en chaque position les pondérations associées à chaque nucléotide. Puis en fixant un seuil à la variable mesurant le poids total du signal reconnu (`p` dans l'exemple 5.7)

avec une contrainte associée à une marque, on détermine la condition de passage de cette marque. Ce passage indique la reconnaissance du signal consensus.

Précisons que pour les chaînes de Markov d'ordre n, nous utilisons d'autres éléments du langage que nous ne détaillons pas dans ce mémoire, contrôlant la progression de la lecture de la séquence. Ces éléments permettent de contrôler un déplacement en arrière de n lettres lors de la lecture. Ceci a pour résultat de définir des scores (ou poids) qui prennent en compte les lettres précédent la lettre courante pour calculer son score.

De plus, afin de faciliter l'utilisation de cette méthode, nous avons développé un logiciel de génération automatique de programme TAGCC pour reconnaissance de signaux consensus en utilisant des chaînes de Markov d'ordre n. Ce programme, nommé TAGLEARN [Dje02] produit automatiquement un programme TAGCC pour un ordre de la chaîne de Markov arbitrairement fixé par l'utilisateur et en fonction d'un échantillon de signaux consensus à analyser. Nous avons encadré S. Djebali (DEA AMIB) dans ce stage. S. Dejbali poursuit actuellement une thèse en co-encadrement (LAMI,CNS,ENS Ulm).

5.4.5 Lecture non déterministe

A chaque étape, le lecteur effectue un traitement de reconnaissance et de calcul. Lors du passage d'une transition, une transition marquée constitue une alternative possible à la reconnaissance faite en utilisant la transition étiquetée de la lettre courante. Dans ce cas, le lecteur évalue les conditions associées à la transition marquées et emprunte celle-ci. Il empruntera aussi bien sûr la transition associé à la lettre lue. Ce principe permet d'avoir une lecture non déterministe de la séquence, rendant plusieurs lectures possibles. Chaque lecture valide se distingue d'une autre par la position des marques sur la séquence étudiées, constituant autant d'interprétations possibles de la séquence en fonction de la spécification du programme. L'ordre dans lequel vont être franchies les transitions, à savoir lire la lettre puis lire la marque ou inversement correspond à une stratégie particulière de lecture. La première stratégie se nomme ASAP (As Soon As Possible) et la seconde ALAP (As Late As Possible).

Celle-ci influe uniquement sur l'ordre d'énumération des solutions. Schématiquement, la stratégie ASAP insère en premier toutes les marques possibles et le nombre de lettres séparant deux séquences débute en étant le plus petit possible. La stratégie ALAP fait l'inverse. Si l'on considère l'automate de la figure 5.6 correspondant à l'équation `relation=A*/{MARK}(A:{l=1})*` la stratégie ALAP énumérera les solutions pour l dans l'ordre décroissant alors que la stratégie ASAP les énumérera dans l'ordre croissant. L'espace des solutions se présente sous la forme d'un arbre binaire où les noeuds indiquent les points de choix et les arcs les choix opérés. La figure 5.8 décrit cet espace pour la séquence `AAA`. La lecture de la marque correspond à l'arc de gauche, celle de la lettre celui de droite lorsque un choix doit s'effectuer. La stratégie ASAP correspond à un parcours gauche à droite de cet arbre alors que la stratégie ALAP correspond au parcours inverse.

Le programmeur déterminera pour chaque marque la stratégie qu'il souhaite utiliser afin d'influencer l'ordre d'énumération. Syntaxiquement, la stratégie ASAP, stratégie par défaut, correspond à la flèche `->` alors que la stratégie ALAP s'obtient en utilisant la flèche `=>`. Par exemple, `relation=A*/{true ->MARK}(A:{l=1})*` exprime explicitement l'emploi de la stratégie ASAP alors que `relation=A*/{true =>MARK}(A:{l=1})*` celle ALAP.

Le passage de la marque étant sujet à aucune condition, `true` est insérée. Ces stratégies permettent d'obtenir en premier des solutions jugées heuristiquement plus intéressantes que d'autres et

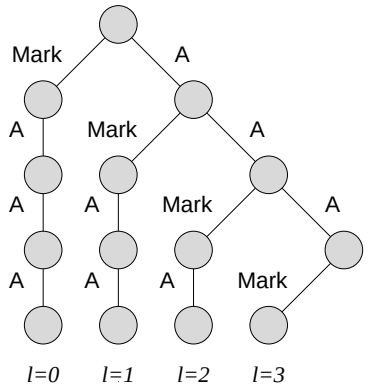


FIG. 5.8 – Représentation de l'espace des solutions pour la reconnaissance de `AAA` en fonction de l'automate pondéré marqué de la figure 5.6.

d'interrompre le traitement avant son terme car comme nous le verrons dans la section 5.6 ce traitement peut donner lieu à un nombre exponentiel de solutions. Choisir une stratégie conduit donc à accélérer la convergence du traitement vers des solutions jugées plus intéressantes que d'autres.

A titre d'exemple, on peut remarquer que les solutions pour les introns sont parmi les séquences les plus longues qui sont proposées et les exons parmi les plus courtes. Aussi, pour forcer l'énumération à débuter par des solutions présentant de longs introns et de courts exons, on appliquera la stratégie ALAP à une marque identifiant la fin d'un intron, et une stratégie ASAP à une marque définissant la fin d'un exon (début d'un intron). Il s'agit bien sûr d'une heuristique «biologique» sans autre validation qu'une comparaison avec des séquences-tests.

5.5 Expressivité et efficacité

Dans cette section nous résumons les éléments de validation concernant l'idée que la spécialisation constitue un moyen de concilier expressivité et efficacité.

Le choix des relations rationnelles comme base théorique aux développement de ce langage ne se réduit pas au pouvoir expressif qu'elles procurent. Il est aussi motivé par un souhait de produire des programmes efficaces. Cette efficacité a été déjà mise en valeur par M.Mohri [MPR00, Moh97] dans le cadre de la reconnaissance de la parole où la performance d'un programme constitue un élément critique pour obtenir une analyse en temps réel. Dans le domaine de l'annotation, le besoin d'efficacité se manifeste lors du traitement sur un grand nombre de séquences. A ce sujet, l'utilisation des relations rationnelles et surtout du processus de reconnaissance/calcul permet de produire un programme qui possède la meilleure complexité pour des traitements routiniers de la prédiction de gènes. Schématiquement, ces traitements routiniers correspondent à l'attribution d'un poids à des mots particuliers. Ce traitement se trouve par exemple être au cœur des mesures par contenu fondées sur un modèle d'ADN codant tel que l'usage de codons [RG00].

Il s'agit donc d'implanter une fonction totale des mots vers un domaine de valeurs qui correspond à une table associative. La recherche la plus rapide (en complexité) est obtenue en utilisant des automates pondérés. Elle est indépendante du nombre de signaux. La table 5.2 rappelle les

Méthode	Complexité au pire	Commentaire
Table non-triée	$O(M.n)$	Algorithme «Force Brute»
Table triée	$O(n.\log(M))$	
Table de hachage	$O(M.n)$	La complexité intéressante, en moyenne, est en $O(n)$.
Automate pondéré	$O(n)$	implantation équivalente à celle d'un dictionnaire.

TAB. 5.2 – Complexité des méthodes de recherche dans une table associative de M mots de taille n

Langage	Complexité de la Reconnaissance	Problème en Biologie
Rationnel, Regular	linéaire	reconnaissance de la structure primaire, reconnaissance de signaux.
Algébrique, Context free	polynomial , $O(n^3)$ en général, $O(n^2)$ si la grammaire n'est pas ambigue, $O(n)$ si la grammaire est $LALR, LR, LL$	structure secondaire de l'ARN, palindrome.
Contextuel, Context sensitive	exponentiel	structure secondaire de l'ARN avec pseudo-nœud
Récursivement énumérable, Recursively enumerable	indécidable	

cette table résumé les différentes classes de langages.
- La première colonne décrit les catégorie de langages.
- La seconde colonne donne la classe de complexité de l'algorithme de reconnaissance
- La troisième colonne rapproche ces classes d'application de reconnaissance de séquences en biologie.

TAB. 5.3 – Hiérarchie de Chomsky & Problèmes en génomique (d'après [Str01], voir aussi [Sea02])

complexités de différentes méthodes de recherches dans une table associative.

La table 5.3 reprend la classification des langages proposée par Chomsky en lui associant des problèmes biologiques caractéristiques. Cette table attribue aussi à chaque classe de langage, la classe de complexité de l'algorithme reconnaissance d'un mot. Comme on peut le constater la complexité de la reconnaissance augmente en fonction de l'expressivité de la grammaire. Définir la plus petite classe de langage permettant de couvrir un problème permet en ce sens de définir le programme le plus performant pour la reconnaissance. La reconnaissance en TAGCC s'effectue de manière linéaire pour les cas les plus simples.

Une étude préliminaire montre qu'une application écrite en TAGCC, nommé PROMTAG, recherchant des promoteurs [Dje02], permet de rédiger des programmes performants de manière concise.

Le nombre de solutions et le temps d'exécution d'un programme TAGCC dépend bien sûr de la nature du problème qu'il résoud. Certains programmes comme l'identification d'un signal possèdent une complexité linéaires tandis que d'autres sont exponentiels. Par exemple, si l'on considère l'équation suivante : `equ=N*/{M1}N*/{M2}...N*/{Mi}..N*/{Mq}` qui correspond à une segmentation de la séquence en q sous-séquences, on démontre [Del01] que le nombre de solutions pour une séquence de

longueur n est C_{q-1}^{n-1}. Cette équation décrit d'une manière caricaturale une analyse insuffisamment discriminante. Par exemple, l'utilisation des signaux AG et GT (cf. figure 5.1) présents dans les sites donneurs et les sites accepteurs d'épissage est insuffisante pour séparer les introns des exons. Le nombre de solutions respectant cette contrainte reste exponentiel. L'annotation demeure donc un problème difficile malgré le cadre plus expressif et plus efficace pour les traitements routiniers offert par TAGCC.

5.6 Etat de l'art et comparaison

Dans cette section nous établirons une comparaison de TAGCC vis à vis des autres langages dédiés à l'analyse des séquences biologiques. La conception de langages spécifiques au domaine de l'annotation fut initié par D. Searls qui proposa le langage GENLANG. A notre connaissance, D. Searls fut aussi l'initiateur de l'idée d'utiliser la capacité à décrire la structure d'un gène par une grammaire afin d'effectuer des prédictions. Pionner en ce domaine, son auteur fonde son formalisme sur les *String Variable Grammar* [Sea93, Sea94, DS94]. Il permet de décrire de manière concise la structure grammaticale d'un gène. Ce langage est basée sur PROLOG. Intrinsèquement, la reconnaissance d'un gène par la seule identification de sa structure conduit actuellement à un indéterminisme quant à l'interprétation des signaux et du choix des règles. Cet élément a été aussi mis en valeur lors de la présentation de TAGCC dans la section précédente. L'indéterminisme se gère dans GENLANG par un mécanisme additionnel de sélection d'une règle de grammaire par rapport à une fonction locale de coûts à minimiser et à des contraintes à satisfaire. Plus précisément, la grammaire décrites par GENLANG possède des règles ambiguës. La sélection d'une règles plutôt qu'une autre pour reconnaître un mot s'effectue par rapport à une fonction objective de coût calculant la sélection de la règle de meilleur coût.

Les fonctionnalités de GENLANG et de TAGCC ne se recouvrent pas totalement. En se restreignant strictement au langage, le formalisme SVG de GENLANG contient strictement celui des langages algébriques (context-free), eux-mêmes contenant strictement les langages rationnels (cf. 5.3). Sa conception a été motivée par la la recherche de structures secondaires de l'ARN où la détection de palindrome joue un rôle central [FTR02]. La reconnaissance de palindromes ne peut être codée dans un langage rationnel. Remarquons que la description proposée par D.Searls [Sea94] de la structure grammaticale d'un gène se décrit aussi en TAGCC (programme 5.10). L'intérêt des relations rationnelles réside dans la définition d'un cadre unifié pour programmer différentes méthodes de prédiction fondées sur les matrices de poids, les chaînes de Markov, sur le respect de longueurs minimales séparant des signaux, etc. Ces méthodes se formalisant par des relations rationnelles particulières, elles s'écrivent en TAGCC. En se fondant uniquement sur le formalisme SVG, il apparaît clairement que ces méthodes ne peuvent être décrites en GENLANG car celui-ci restreint la description à une structure grammaticale. Dans GENLANG, des macro-opérateurs spécifiques effectuant la reconnaissance de signaux consensus doivent être ajoutés en plus du formalisme SVG pour effectuer des opérations de calcul sur les mots. Ces macro-opérateurs faisant appel à des bibliothèques, il n'entrent pas dans le cadre du formalisme SVG alors qu'en TAGCC, le calcul fait partie du formalisme. Bien que GENLANG possède un pouvoir de description de grammaire supérieur à celui de TAGCC, son formalisme ne peut décrire directement certaines méthodes qui calculent des scores à partir les mots telles que celles fondées sur la discrimination de signaux consensus à partir des chaînes de Markov.

Les langages spécifiques sont aussi utilisés pour l'analyse des séquences protéiques. PROSITE est un langage déclaratif permettant de décrire des parties de séquences protéiques. PROSITE est à la fois un langage de requête et un langage de description de séquences. Comparativement à ce

dernier, TAGCC offre les extensions des relations rationnelles face aux expressions rationnelles. De plus, grâce au processus de marque conditionnelle, il étend le processus de reconnaissance en intégrant des conditions. En résumé comparativement à PROSITE, TAGCC permet de calculer différentes valeurs au fur et à mesure que la reconnaissance s'effectue, puis d'appliquer des contraintes sur les valeurs calculées et enfin il offre un mécanisme d'énumération des différentes solutions possibles. Il faut cependant préciser que PROSITE a été principalement conçu pour décrire des bases de données de motifs protéiques plutôt que de décrire des requêtes permettant d'interroger ces bases, ou encore d'annoter un génome. En ce sens, les éléments composant PROSITE offrent clairement une expressivité suffisante pour remplir leur rôle de description des motifs complexes.

5.7 Extension

Extension de la résolution : vers une résolution sélective La résolution actuelle énumère la totalité des solutions dans l'ordre donné par les stratégies de passage de marque. Ces solutions sont contraintes à la fois par la description des signaux et par des conditions définies pour le passage de marques. La résolution a donc essentiellement pour rôle de discriminer parmi les propositions les solutions respectant les contraintes inhérentes au langage et celles inhérentes aux quantités calculées. Ce mécanisme permet l'implantation de méthodes dont l'objet est d'opérer une sélection entre les solutions retenues et les autres. Cependant, la validation du langage par l'implantation de méthodes d'annotation a mis en évidence l'intérêt de posséder un mécanisme additionnel plus sophistiqué que celui dépendant des stratégies ASAP et ALAP. Ce mécanisme repose sur la possibilité d'ordonner les solutions par rapport à une fonction objective de coût qui attribuerait un score représentant la qualité d'une solution. Cette fonction est définie à partir des variables calculées lors de la reconnaissance.

Dans cette perspective, il est alors possible de ne retenir qu'un sous-ensemble de solutions possibles possédant les meilleurs scores. Dans les logiciels d'annotation utilisant ce procédé, le score représente un indicateur de confiance en une solution. Il est défini par des règles heuristiques. Les premières études de cette extension, entreprises avec A. Dubas [Dub02] au cours de son D.E.A, mettent en valeur deux classes de problème dépendant respectivement : de la représentation des solutions à adopter pour retenir les meilleures solutions, et de la résolution en elle-même.

Afin de sélectionner des solutions parmi toutes les solutions possibles, il est nécessaire de disposer d'une représentation *finie* décrivant l'espace complet des solutions dans lequel on extrait celles de meilleures scores.

La représentation retenue est un graphe acyclique valué orienté, appelé graphe marque/position, où chaque nœud représente le placement d'une marque à une position possible. Les arcs sont valués grâce à une fonction de coûts. Deux nœuds sont reliés entre eux si la reconnaissance de la séquence permet en partant d'une marque placée à une position donnée d'atteindre la prochaine marque. Cette reconnaissance se réalise sans évaluer les conditions ce qui permet de construire efficacement cette représentation à partir d'une séquence. Dans ce graphe, une solution correspond à un chemin reliant la première marque à la dernière.

En conséquence, la résolution pour obtenir un certain nombre de solutions de meilleurs coûts revient donc à un *problème de recherche de k plus courts chemins dans un graphe acyclique orienté.* Nous avons adapter un algorithme de recherche de plus courts chemins en collaboration avec M. Manceny lors de son stage de D.E.A.

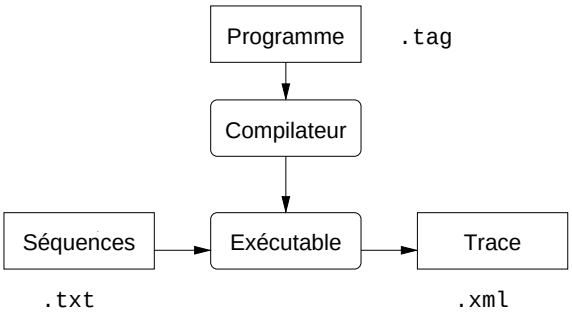
Extension du langage : vers une annotation intégrative La variété des programmes d'annotation reflète la multiplicité des modèles définissant la structure d'un gène. Faire inter-agir ces derniers de sorte à améliorer la qualité de l'annotation conduit naturellement à l'étude des moyens de combinaison de programmes. Combiner des programmes couvrent un large spectre de domaines et de méthodes informatiques. Deux approches peuvent être considérées pour combiner les résultats des programmes : La première concerne la combinaison de différentes modélisations, parcelles d'un modèle global, et dont la réunion couvre la modélisation complète de l'objet observé. Chaque modèle agit en complémentarité d'autres en se focalisant sur une particularité telle, par exemple, que la recherche du premier exon. Dans ce cas le logiciel intègre les différents éléments. Au contraire, la seconde s'appuie sur la différence et parfois peut-être l'opposition des modèles qui reflètent une autre forme de complémentarité celles des «points de vues». La confiance d'une solution dépend alors du nombre de logiciels l'ayant trouvée.

Par delà l'amélioration apportée par la comparaison de programmes d'annotations, cette approche permettrait de distinguer les parties génériques des programmes des parties contingentes à un génome. Cette distinction permettrait de connaître les particularités liées à une classe de gènes, comme par exemple ceux liés à un génome d'un organisme Ceci affine la représentation de la structure d'un gène en distinguant des éléments propre à une classe d'éléments plus génériques.

Les annotateurs humains réalisent la synthèse des analyses des résultats de logiciels d'annotations. L'automatisation de cette analyse conduit à informatiser cette expertise. Cette extension du travail sur l'annotation automatique débuté dans le cadre du D.E.A. de Sarah Djebali [Dje02] se poursuit actuellement en thèse en co-encadrement avec Hughes Roest Crollius (CNS, ENS Ulm). Des travaux [MT98, KHD02] ont déjà été proposés en ce sens. Cependant l'orientation prise de ce projet se distingue de ceux mentionnés par le fait qu'il s'agit de définir un programme exprimant qualitativement les choix opérés. Ceci conduit à formaliser les critères d'un expert humain qui conduisent au choix d'une solution plutôt qu'une autre,

5.8 Bilan

TAGCC est un langage spécialisé au domaine de l'annotation dont l'objet est de décrire des structures de gènes comme se décrivent les structures grammaticales des langages. Il offre un cadre de programmation pour la résolution de la prédiction des gènes. Pour permettre une approche informatique nous avons caractérisé la problématique informatique sous-jacent par la définition d'un problème qui se nomme S^2CP. Ce problème met en valeur la nécessité de coupler un calcul à une reconnaissance de signaux. Le mécanisme interne permettant de combiner reconnaissance et calcul se base sur l'emploi d'automates pondérés dont l'efficacité a été démontrée dans les domaines de la linguistique. Ce mécanisme permet, tout en offrant une expressivité certaine, une résolution très efficace. L'originalité technique de ce langage concerne la définition d'un mécanisme unique permettant d'annoter des régions fonctionnelles d'intérêt dans une séquence, de gérer les solutions alternatives et enfin de prendre en compte le dernier aspect du problème S^2CP que sont les conditions. Ce mécanisme se fonde sur la notion de marque avec contrainte. Une marque sera reconnue uniquement si la condition associée est satisfaite. Dans ce cas la séquence se verra marquée à la position où cette reconnaissance s'applique. Ce marquage offre un procédé pour *« souligner »* les zones présentant un intérêt fonctionnel. Les solutions alternatives sont trouvées en combinant différemment les marques. Bien qu'une marque puisse être reconnue, le lecteur considérera à chaque fois que cela est possible la transition conduisant au marquage et celle conduisant à la lecture classique d'un caractère.



L'architecture de TAGCC s'organise autour d'un compilateur qui transforme un programme TAGCC en un automate pondéré. Un exécutable contient la description d'un automate pondéré marqué, un lecteur et des fonctionnalités permettant de lire un fichier selon des formats différents et d'écrire la trace de la reconnaissance sous forme de fichiers `html` qui peuvent être lus par un navigateur.

FIG. 5.9 – Description de l'environnement TAGCC

Débuté en 2001, le prototype de compilateur du langage TAGCC a été écrit en OCAML . La version actuelle (1.2) comporte 7000 lignes. Il est accessible gratuitement par Internet [Tag]. L'architecture du logiciel est la suivante : Un programme TAGCC contient l'ensemble des équations définissant la structure d'un gène. Ce programme est compilé pour produire un exécutable. Le source de cet exécutable correspond à un programme OCAML contenant l'automate pondéré marqué et le lecteur. En complément de ces éléments composants le cœur du programme, des fonctionnalités se rapportant aux interfaces sont intégrées. Elles concernent la lecture d'une séquence selon différents formats (fasta, gcg, embl, genbank, etc.), la sortie sous forme XML (html), et d'autres fonctions secondaires tel que le calcul de la séquence complémentaire. La figure 5.9 décrit l'architecture générale de l'environnement TAGCC.

Les programmes actuels développés en TAGCC correspondent à des parties de programmes d'annotations. En ce sens, ils valident l'adéquation de ce langage au domaine de la prédiction *ab-initio*. La maturité du logiciel nous permet à présent d'envisager de développer un annotateur réel. La collaboration entreprise avec le C.N.S nous permettra d'avancer dans cet objectif.

Sur ce projet, j'ai encadré quatre étudiants de DEA. L. Rival [Riv01] (DEA AMIB) s'est occupée de l'implantation d'un programme d'annotation TAGCC qui validait les constructions du langage. S. Djebali [Dje02] (DEA AMIB) s'est concentrée sur un générateur automatique de programme TAGCC à partir d'analyse de signaux. A. L. Dubas [Dub02] (DEA Informatique) en 2002 puis M. Manceny (DEA AMIB - école d'ingénieur IEE) en 2003 ont contribué au développement d'un nouveau mécanisme de résolution qui permet de sélectionner des solutions les plus probables face à l'ensemble des solutions possibles.

TAGCC a été originellement le fruit d'une collaboration avec G. Waksman et J. Lamartine du laboratoire de génétique le LGR. Actuellement, une collaboration s'est engagée avec le Centre National de Séquençage et l'Ecole Normale Supérieure (Ulm), dans le cadre de la thèse de S. Djebali [DDC03], pour permettre l'intégration de programmes d'annotations différents.

```
1   // Définition élémentaire ---
    N  = [A T G C]; Pu = [A G]; Py = [T C];

    // Code génétique ---
5   Ala = G C N;  Arg = [(C G N)  (A G A) (A G G)] ; Asn = A A Py ; Asp = G A Py;
    Cys = T G Py; Gln = C A Pu                     ; Glu = G A Pu ; Gly = G G N;
    His = C A Py; Ile = A T [ A T C ]              ; Lys = A A Pu ; Leu = [(T T Pu) (C T N)];
    Met = A T G ; Phe = T T Py                     ; Pro = C C N  ; Ser = [(T C N) (A G Py)];
    Thr = A C N ; Trp = T G G                      ; Tyr = T A Py ; Val = G T N;
10
    // Signaux et codon ---
    Start = Met;
    Codon = [Ala Arg Asn Asp Cys Gln Glu Gly His Ile Lys Leu Met Phe Pro Ser Thr Trp Tyr Val];
    Stop = T [(A Pu)  (G A)] ;
15  // exon ---
    nat len; seq gene;
    a=A:{len=1,gene=A}; t=T:{len=1,gene=T}; g=G:{len=1,gene=G}; c=C:{len=1,gene=C};
    exon      = N*{0:2} Codon+ N*{0:2} \{(A,a) (T,t) (G,g) (C,c)} ;
    exon_last = N*{0:2} Codon+         \{(A,a) (T,t) (G,g) (C,c)} ;
20  // Intron ---
    Donor    = G T; Branch = A; Acceptor = A G;
    Intron = Donor  N+ Branch N+ Acceptor;
    // Sequence ---
    utr5 = N*;  utr3 = N* ;
25  sequence  =
          N* utr5
          /{BEG_ORF}
               Start (  exon  /{DONOR}  Intron   /{ACCEPTOR} )* /{LAST_EXON}  exon_last  Stop
          /{(len%3==0) &&  (gene ?? (Codon+)) -> END_ORF}
30        utr3 N*;
    sequence;
```

Ce programme décrit la grammaire élémentaire d'un gène en fonction des signaux durs. Les définitions sont classiques et reprennent la structure décrite à la figure 5.1. Cette note précise plus spécifiquement le rôle de certaines équations. Les premières équations définissent les éléments fondamentaux : purine, pyrimidine, codon, etc. Les équations `a,t,g,c` (ligne 19) définissent les nucléotides en leur attribuant des valeurs pour calculer la longueur et le gène épissé (`len,gene`). Les modifications doivent être reportées dans tous les exons. Elles s'effectuent grâce à la substitution inscrite en queue des équations définissant les exons (`exon,exon_last`). Le décalage de phase possible est représenté par l'expression `n*{0 :2}`. Les équations correspondant aux régions non-transcrites (`utr5,utr3`) sont inutiles pour ce programme car elles ne diffèrent pas d'une succession de nucléotides. L'équation `sequence` est l'équation principale. Pour tester si le gène est en phase, on utilise un opérateur `??` dont le résultat est un booléen. Celui-ci est vrai si le terme de gauche appartient au langage de droite décrit par une expression rationnelle. Pour accélérer le traitement, on vérifie au préalable si le nombre de nucléotides est un multiple de 3 (`len%3==0`). Cette condition est redondante par rapport à la seconde et sert à restreindre le nombre d'appels à la reconnaissance d'un gène en phase. Tout gène est reconnu par ce programme. Toutefois, il nécessite d'autres tests car le nombre de solutions qu'il propose est très grand.

FIG. 5.10 – (prog.) Le noyau d'un programme d'annotation de gènes

Chapitre 6

Modélisation des réseaux de régulation biologique

Ce chapitre décrit des travaux en cours sur la modélisation des réseaux biologiques. Son principal objectif est d'établir un cadre de recherche de travaux futurs en s'appuyant sur ceux déjà réalisés.

6.1 Introduction

La biologie moderne s'est fixée comme objectif d'expliquer les systèmes biologiques par des phénomènes biochimiques à l'échelle moléculaire. Cette démarche conduit à la fois à un processus de désassemblage des systèmes jusqu'à leurs composants moléculaires et à un assemblage par construction des systèmes biologiques au sein desquels ces constituants exercent une fonction. L'arrivée massive des données du génome, du transcriptome et du protéome, offre l'opportunité d'étudier qualitativement et quantitativement ces mécanismes.

Bien qu'essentiel, l'inventaire des gènes d'un génome ne suffit pas à cerner le comportement d'un organisme car celui-ci dépend aussi des interactions protéiques et géniques. Son étude implique d'élargir le champ d'analyse d'un génome à son environnement moléculaire afin de prendre en compte les interactions qui gouvernent la dynamique d'expression des gènes. Cet élargissement s'appuie sur l'analyse des données du génome, du transcriptome et du protéome.

Cependant, cette analyse révèle tout d'abord la complexité des systèmes biologiques, pas tant par la masse de données, ni par la variété des interactions, mais surtout par la difficulté à les interpréter en regard *des fonctions cellulaires*, c'est à dire des processus identifiés de l'activité de la cellule. La complexité se présente alors comme l'un des objets centraux d'étude de la biologie moderne dont la compréhension établirait le lien entre le comportement d'un organisme et les interactions

moléculaires. Dans ce cadre, l'arrivée massive des données biologiques *pose de manière centrale une question de méthode* pour l'étude de la complexité qu'elles révèlent.

Le développement de modèles et de méthodes est toutefois soumis à la particularité que les mécanismes de réactions biochimiques demeurent partiellement inconnus. Il est difficile d'obtenir des informations quantitatives sur les paramètres des cinétiques et des concentrations moléculaires[1]. De plus, d'autres phénomènes, comme la compartimentation cellulaire, apparaissent jouer un rôle majeur mal dominé. Les modélisations numériques rencontrées dans d'autres disciplines ne peuvent s'appliquer sans extrapoler la valeur numérique de certains paramètres. Plutôt que de trouver des modèles numériques, on cherchera donc à qualifier les interactions entre gènes.

Pour décrire les interactions, les graphes offrent une abstraction permettant de décrire de façon homogène des composants biologiques de nature différente [ZO02]. Il est ainsi possible de développer des modèles couplant protéines et gènes, voies de signalisation et réseau de régulation. Par exemple, pour les réseaux de régulations génétiques, chaque sommet est le représentant d'un gène et de son activité[2]. Les connexions incarnent une régulation d'un gène, exercée par un autre gène [Tho91, Kau93].

Sous cette forme, un système biologique se présente comme un réseau de composants distribués hétérogènes coopérants. Chaque composant réalise, de manière locale, une tâche qui concourt à la réalisation d'une activité globale. L'analyse s'applique à deux niveaux :

- à un niveau local où l'on définit le comportement propre du système. Pour l'étude des réseaux génétiques, cette analyse renseigne sur l'activité d'un gène en fonction de ses régulateurs. Le système est étudié isolément afin d'en définir les caractéristiques propre à sa dynamique. Elles serviront à identifier sa fonction.
- à un niveau global où l'on définit son comportement dans un système intégré. Cette analyse permet de compléter la précédente en indiquant son action sur les autres composantes du réseau.

La complexité réside alors dans l'interprétation du comportement du processus global en fonction de composants connectés. En incarnant les interactions entre composants biologiques, la complexité de la topologie du réseau et de sa dynamique s'assimile en partie à celle du système biologique.

L'étude de la topologie vise à révéler *des principes de conception* [Kit02, SSOA02] qui régissent la régulation. Cette approche porte sur l'analyse de sous-réseaux qui exhibent des propriétés caractéristiques de régulation. Par exemple, un cycle dans un réseau génétique correspond à un phénomène de régulation spécifique [TSRT95]. La topologie permet d'inférer certains éléments de la dynamique de régulation à partir de sous structures du réseau.

Plus généralement, c'est la compréhension des phénomènes d'intégration des réseaux qui est visée. Elle se fonde sur l'étude systématique de sous-structures caractéristiques qui représentent des supports des phénomènes de régulation. L'interconnexion devient l'un des objets d'explication de la régulation qu'exercent les gènes sur l'organisme car elle apparaît en incarner la fonctionnalité. Nous nommerons celle-ci, le *contrôle* afin de la distinguer de la fonction en biologie qui s'applique aux fonctions biochimiques.

La déduction de propriétés liées à la dynamique d'un système biologique à partir du réseau

[1] La concentration est une mesure exprimant le nombre de mole par unité de volume. Dans le contexte de ce chapitre, elle exprime une quantité d'un produit du gène (ARN ou protéine).

[2] L'activité est une grandeur introduite par G. N. Lewis pour exprimer les propriétés thermodynamiques des solutions. Schématiquement, l'activité d'une entité chimique réagissant traduit sa disponibilité pour la réaction considérée. Dans le contexte de ce chapitre, l'activité d'un gène exprime sa capacité de régulation.

peut s'appuyer sur la simulation. Nous avons développé en collaboration avec JL. Giavitto et O. Michel un simulateur pour un réseau de régulation [GMD01, JLGD03] en utilisant le langage $8\frac{1}{2}$ [Mic96, GM02].

On peut aussi utiliser une analyse statique, à la fois de la topologie du réseau de régulation et des informations attribuées à chaque connexion, afin d'extraire des invariants de la régulation qu'il exerce. Son étude met en avant la notion de *module* qui fait référence à une partie du réseau à laquelle s'associe un contrôle particulier. Chaque module incarnerait un bloc élémentaire caractéristique de la construction du réseau [Del02]. Dans ce cadre, l'étude de la régulation s'articule selon deux principes : le premier correspond à l'identification de modules élémentaires caractéristiques de phénomène de régulation, le second à leur assemblage pour engendrer des comportements complexes de régulation.

La modularité introduit en même temps un autre principe, celui d'une *architecture d'intégration*. En effet, en complément naturel à la distinction de sous réseaux (modules), l'intégration vise à donner un explication à des fonctionnalités plus complexes. Chaque sous réseau élémentaire s'assemble pour former un réseau qui peut à son tour être considéré comme étant le support d'une fonctionnalité partielle mais plus intégrée. Cette architecture offre une structure d'organisation de la complexité d'un système biologique par inclusion de sous systèmes dans un système plus complet du point de vue des fonctionnalités.

Les interactions géniques ainsi décrites sous la forme de réseaux de processus concurrents coopératifs et intégrés de manière modulaire posent les bases d'une analyse qui présente des caractéristiques communes avec les réseaux de processus concurrents en informatique. Cette analyse peut se développer sur des théories utilisées pour modéliser les systèmes informatiques concurrents telles que les réseaux de Pétri [MDNM02], les algèbres de processus [DC03] ou les automates [BCC+03]. L'automatisation de son étude par des logiciels conduit à adopter une approche similaire à celle menée pour les systèmes informatiques. Ce rapprochement définit un cadre de modélisation pour la régulation similaire à celui proposé pour confronter un programme à son exécution.

Les modèles informatiques définis pour cette confrontation en compilation, spécification, test de logiciels ... développent une approche discrète et qualitative. Cette modélisation est suffisante pour arriver à des conclusions fiables. Elle n'a généralement pas besoin d'une analyse plus quantitative. En effet, son objectif est de capturer la logique interne du programme afin d'en révéler, par exemple, les incohérences, les "bogues", les optimisations... La modélisation qualitative peut par exemple révéler une certaine forme de pathologie liée au dysfonctionnement du contrôle du réseau génétique. L'intérêt d'un modèle qualitatif dépasse ici l'argument d'une adéquation existant avec les possibilités expérimentales car elle permet de caractériser certains dysfonctionnements liés à la régulation.

Cette étude repose en premier lieu sur la définition d'une sémantique qui permet d'interpréter les informations du réseau en regard de la dynamique. Pour les réseaux génétiques, elle détermine l'évolution de l'état des variables représentant des concentrations de produits du gène, soit l'ARN, soit les protéines.

Dans l'équipe BioInfo du LaMI, nous nous sommes essentiellement concentré sur l'étude des réseaux de régulation en étendant la théorie de René Thomas. En collaboration avec l'équipe de vérification des systèmes embarqués de l'IRCYN, nous avons proposé une sémantique [BCC+03] pour la régulation des réseaux génétiques étendant la théorie de René Thomas.

Nous débuterons par un exposé des travaux sur la dynamique des réseaux car ils permettent d'établir la relation entre réseau et régulation (section 6.2). Nous nous appuierons sur cette dynamique pour exposer les travaux entrepris sur l'analyse statique des réseaux de régulation. Nous examinerons dans la section 6.3 deux catégories d'applications qui montrent deux facettes d'une au-

tomatisation possible de la modélisation sur les réseaux de régulation : la validation et l'inférence de réseau. La validation consiste à confronter le modèle aux propriétés attendues. L'inférence consiste à construire un réseau correspondant aux résultats d'expériences.

Leur automatisation doit tenir compte de deux paramètres : la complexité des systèmes biologiques et la masse des données issues de ces systèmes. Tous deux contribuent naturellement à augmenter la complexité (de calcul) des algorithmes. Pour rendre cette approche efficace, il est nécessaire d'adopter une stratégie incrémentale de construction automatique ou de validation du réseau. Cette stratégie suppose d'adopter une approche se fondant sur la modularité appliquée au système biologique qui permet de partir des composants jusqu'au réseau totalement assemblé. Nous aborderons dans la section 6.4 nos premiers travaux examinant la modularité des réseaux de régulation.

6.2 Dynamique des réseaux de régulation et sémantique

L'objet d'une formalisation est de fournir un cadre sur lequel il est possible de réaliser des logiciels et de valider *automatiquement* des propriétés sur le système. Il est à distinguer de l'approche de modélisation en vue d'expertise humaine. Il s'agit de définir un langage formel dans lequel l'expert exprime son étude dans une perspective de traitements automatiques. Par soucis de simplicité, nous présenterons dans l'exposé une version simplifiée de la sémantique fondée sur les réseaux booléens. Nous renvoyons le lecteur à [dJ02] pour un résumé et une bibliographie des différentes modélisations proposées que l'on complétera par les travaux de [CD02a, CD02b] sur le rôle des boucles de régulation.

Dans les années 1970, le généticien René Thomas proposa un formalisme pour modéliser l'expression des gènes [TTK95, TT95]. Les réseaux de régulation génétiques décrivent un ensemble d'équations qui détermine l'évolution d'un état se rapportant à l'expression des gènes. René Thomas développa sa théorie sur *les réseaux booléens* et en proposa une généralisation pour décrire des espaces d'états à valeur entières.

Ce modèle présente les caractéristiques d'être discret et asynchrone. Sous l'action d'une régulation amplifiant la synthèse, *une activation*, la concentration celui-ci évolue selon une sigmoïde. Il en va de même pour une action de régulation réduisant ce taux, *une inhibition*. Une fonction en sigmoïde peut être approchée par une fonction à seuil. Cette approximation fait apparaître des paliers ou niveaux que l'on modélise par des valeurs entières les représentant.

La régulation d'un gène dépend de plusieurs étapes (transcription, traduction, fixation de la protéine sur l'ADN) dont le temps varie pour chaque gène. Ceci conduit à retenir l'hypothèse d'un phénomène asynchrone pour la régulation de plusieurs gènes.

Dans ce modèle, les interactions entre gènes se modélisent par un graphe. Chaque sommet représente un gène. Chaque arc incarne une action régulatrice d'un gène sur un autre. Le gène régulateur est la source de l'arc, le gène régulé la cible. A chaque gène, on associe une variable dont la valeur est la concentration. Dans les réseaux booléens, les valeurs des concentrations sont simplement représentées par des variables booléennes. L'action régulatrice liée à un arc peut être de deux sortes :

- Soit une activation : dans ce cas l'augmentation de la concentration de la variable associée au gène régulateur induira *a priori* une augmentation de la variable associée au gène régulé. Ce couplage s'effectue de même, *a priori*, pour la diminution.

- Soit une inhibition : dans ce cas l'augmentation de la concentration de la variable du gène régulateur induira *a priori* une baisse de la concentration de la variable du gène régulé.

La figure 6.1 décrit les éléments minimaux qui permettent de définir la sémantique des réseaux de régulation booléens. La dynamique d'un réseau génétique se calcule par une fonction de transition. Un réseau génétique est caractérisé par (V, E, Sign) avec V ensemble des sommets, E ensemble des arcs et la fonction de $\mathsf{Sign} : E \mapsto \{+, -\}$ qui associe à chaque arc la nature de la régulation , activatrice $(+)$ ou inhibitrice $(-)$.

Dans une approche asynchrone, une seule variable évolue par transition. Dans ce cas, la fonction transition τ calcule l'évolution d'une variable par rapport à l'état courant. Au sein d'un réseau représentant un individu, le choix d'une variable parmi plusieurs à partir d'un état est *a priori* indéterministe. En biologie, cet indéterminisme s'explique par l'existence de variations entre individus (cellules ou organismes) possédant le même patrimoine génétique (ou tout du moins la même partie modélisée). Ce cas survient notamment dans le développement de l'embryon où des cellules indifférenciées, *les cellules souches*, possédant les mêmes informations génétiques se spécialisent. Ces variations reposent sur des phénomènes *épigénétiques* qui ne dépendent pas seulement des gènes mais aussi de facteurs environnementaux. Ces facteurs interviennent donc indirectement dans le modèle exposé sous forme d'une indétermination dans le choix des variables pour effectuer une transition et dans le choix de l'état initial.

La transition capture l'action des régulateurs sur un gène qui se définissent ainsi : Soit x, une variable, l'ensemble des activateurs (resp. inhibiteurs) se définit ainsi $R_+(x) = \{y \in V | \mathsf{Sign}(x, y) = +\}$ (resp. $R_-(x) = \{y \in V | \mathsf{Sign}(x, y) = -\}$).

Pour une variable x donnée, on spécifie l'évolution pour toute combinaison possible d'activateurs $A(x) \subseteq R_+(x)$ et d'inhibiteurs $I(x) \subseteq R_-(x)$ de x par une valeur notée $K_{x,A(x),I(x)}$ (cf. figure 6.1).

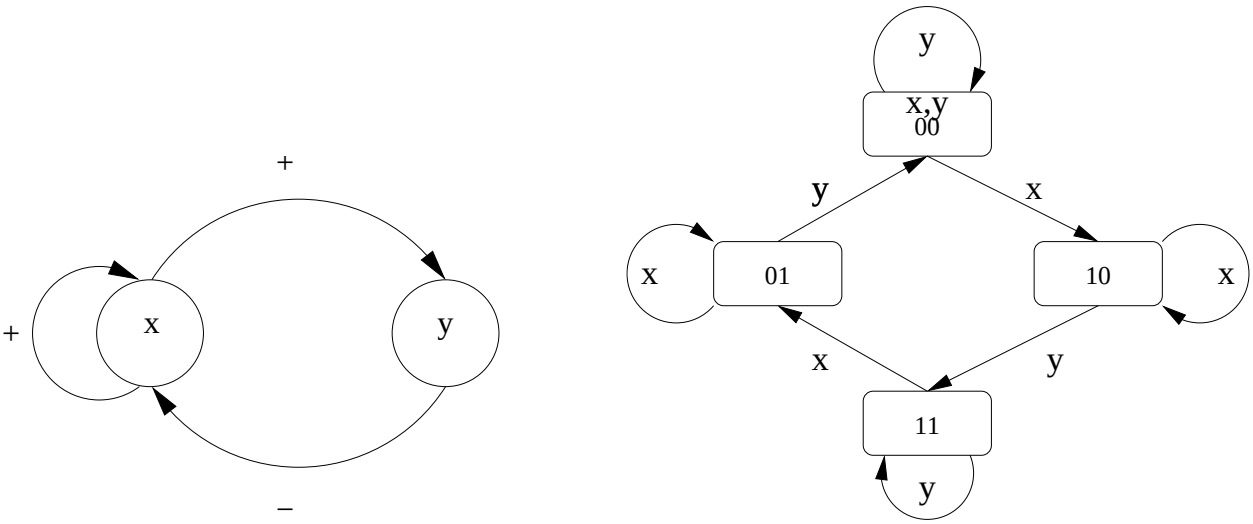
L'évolution d'une variable dépend de régulateurs *effectifs*, c'est à dire dont la valeur est 1. Dans le cas booléen, les paramètres $K_{x,A(x),I(x)}$ déterminent la valeur suivante de la variable x choisie lors de la transition en fonction du plus grand ensemble de régulateurs effectifs. Précisons que dans un cas plus général où les variables prennent des valeurs entières, ces paramètres correspondent à une notion plus complexe d'attracteur pour la variable concernée.

La régulation est décrite par deux éléments : le réseau génétique et les paramètres K. Afin que l'activation et l'inhibition définies dans le réseau représente une augmentation ou une diminution de la concentration définies par les paramètres K, il est nécessaire d'imposer des contraintes sur ces paramètres. Ces propriétés demeurent pour des modèles plus généraux :

- L'*activité* impose qu'au moins une configuration des gènes régulateurs d'un autre gène conduise à un changement d'état du gène régulé. Dans le cas contraire, le niveau du gène reste constant. Aucune modification n'est observée ce qui laisse peu de signification à l'action des régulateurs
- La *monotonie* des paramètres K met en relation la valeur des paramètres et le nombre d'activateurs présents (resp. inhibiteurs). Intuitivement, cette dernière propriété interdit qu'un activateur se comporte comme un inhibiteur ou inversement. Si le nombre d'activateurs augmentent (resp. inhibiteurs) la concentration du gène ne peut baisser (resp. augmenter).

Définition 6.1 (Activité)

$\forall x \in V, \forall y \in R_+(x), \exists X_+ \subseteq R_+(x), \exists X_- \subseteq R_-(x),$
$K_{x,X_+,X_-} < K_{x,X_+\cup\{y\},X_-}$
$\forall x \in V, \forall y \in R_-(x), \exists X_+ \subseteq R_+(x), \exists X_- \subseteq R_-(x),$
$K_{x,X_+,X_-\cup\{y\}} < K_{x,X_+,X_-}$



$$\begin{array}{lllllllll} K_x,\emptyset,\emptyset & = & 1, & K_x,\emptyset,\{y\} & = & 0, & K_y,\emptyset,\emptyset & = & 0 \\ K_x,\emptyset,\{y\} & = & 0, & K_x,\{x\},\{y\} & = & 0, & K_y,\emptyset,\{x\} & = & 0 \\ K_x,\{x\},\emptyset & = & 1. & & & & & & \end{array}$$

Les composants de la sémantique sont :

- *Le réseau de Régulation (en haut à gauche).* Les arcs sont étiquetés par $+, -$, indiquant respectivement l'activation et l'inhibition. Pour ce réseau x est son propre activateur et activateur de y. Par contre y est inhibiteur de x. Soit x, une variable, On définit l'ensemble des activateurs ainsi $R_+(x) = \{y \in V | \mathsf{Sign}(x,y) = +\}$; de même pour les inhibiteurs $R_-(x) = \{y \in V | \mathsf{Sign}(x,y) = -\}$.
- *La table des paramètres K (en bas à gauche).* Pour chacune des variables et en fonction de toutes combinaisons d'activateurs et d'inhibiteurs, on calcule les valeurs des états attracteurs de la variable (paramètres K). Dans le cas booléen, ces états représentent l'étape suivante d'évolution en fonction des activateurs et des inhibiteurs effectifs (ie. possédant une valeur égale à 1).
- *Le graphe d'états (à droite).* Le graphe d'états représente toutes les évolutions asynchrones possibles du réseau. Dans ce graphe, les arcs sont étiquetés par une variable qui désigne la candidate pour évoluer.
- La transition en fonction d'une variable x à l'étape t se définit ainsi : soit (V, E, Sign), un réseau de régulation booléen,

$$\nu_t \xrightarrow{x} \nu_{t+1}, \quad \forall \alpha \in V, \quad \nu_{t+1}(\alpha) = \begin{cases} \mathsf{Si}\ \alpha = x & \nu_{t+1}(x) = K_{x,A(x),I(x)} \\ \quad \text{avec} & A(x) = \{y \in R_+(x) | \nu_t(y) = 1\}, \\ \quad \text{et} & I(x) = \{y \in R_-(x) | \nu_t(y) = 1\}. \\ \quad \text{et} & \forall \beta \in (R_+(x) \cup R_-(x)) - (A(x) \cup I(x)), \nu_t(\beta) = 0 \\ \mathsf{Sinon} & \nu_t(\alpha) \end{cases}$$

  Dans un cas plus général où les variables peuvent admettre des valeurs entières, les paramètres K correspondent à des attracteurs [BCC+03]. Elles désignent alors des valeurs d'expression vers lesquelles les concentations des gènes tendent. Cette valeur limite est atteinte pour un gène si la régulation (ensemble d'activateurs, ensemble d'inhibiteurs du gène) n'est pas modifiée.

Il faut insister sur le fait que la valeur des paramètres est déterminante pour l'évolution du réseau. Par exemple la modification suivante $K_{x,\{x\},\{y\}} = 1$ entraîne les variables du réseau dans un état stable $(1,1)$.

FIG. 6.1 – Réseau de régulation génétique

Définition 6.2 (Monotonie)
$\forall x \in V, \forall X_+ \subseteq R_+(x), \forall X_- \subseteq R_-(x), \forall X'_+ \subseteq R_+(x), \forall X'_- \subseteq R_-(x),$
$X_+ \subseteq X'_+, X'_- \subseteq X_- \Rightarrow K_{x,X_+,X_-} \leq K_{x,X'_+,X'_-}$

Mentionnons l'existence d'une autre catégorie de modèles qui reposent sur des caractéristiques différentes de celles retenues par René Thomas. La figure 6.2 résume les principaux éléments de cette catégorie. Ils sont synchrones et l'évolution des concentrations est définie par une fonction continue. La régulation génétique se modélise alors par un réseau de neurones récurrents (ou avec rétroaction) [MM98, d'H99, WWS99, Voh01a, Voh01b].

La modélisation des réseaux par des réseaux de neurones ou l'approche de René Thomas présentent des caractéristiques différentes. L'absence de mesure quantitative de la cinétique des concentrations sous l'action de régulation rend difficile une comparaison s'établissant sur des critères d'adéquation du modèle à l'expérience. Ils servent à proposer une explication aux phénomènes de régulation. Actuellement, on peut difficilement les considérer comme étant prédictifs.

Tous deux considèrent une évolution du taux de concentration en sigmoïde mais diffèrent sur les points suivants : le domaine valeurs de concentrations (continu ou discret), les règles de transitions dans le temps (synchrone ou asynchrone) et la manière dont est caractérisée l'effet de la régulation (somme pondérée ou attracteur). Dans leur application, les modèles fondés sur les réseaux de neurones sont plutôt utilisés pour la simulation [Voh01a, Voh01b] alors que les modèles inspirés de celui de René Thomas s'orientent plutôt vers l'analyse de la dynamique du réseau. On peut remarquer enfin que les deux catégories de modèles ne prennent pas en compte certaines caractéristique des phénomènes biologiques. Les modèles fondés sur les réseaux de neurones ne tiennent pas compte de l'asynchronisme et de l'indétermination des transitions mais ils conservent une indétermination de l'état initial. Les modèles inspirés de celui de René Thomas considèrent des niveaux discrets pour un phénomène continu. Ce processus d'abstraction qui ne considère pas certains phénomènes est intrinsèque à la démarche de modélisation. La pertinence d'une catégorie de modèles par rapport à une autre repose en partie sur le choix de ces abstractions. Dans l'état actuel des connaissances, il apparaît prématuré de déterminer le modèle le plus adéquat. Cette distinction s'effectuera peut être en fonction de leur pouvoir de prédiction.

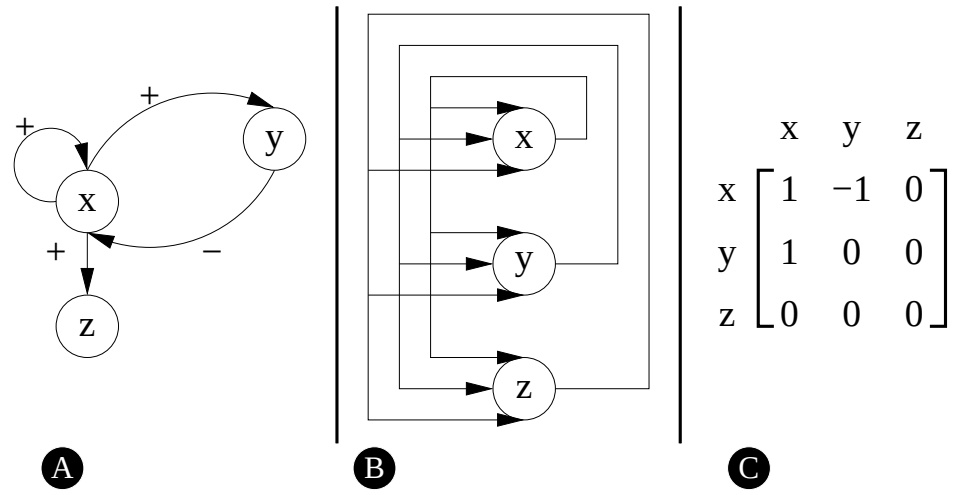
En collaboration avec l'équipe de vérification des systèmes embarqués de l'IRCYN, nous avons défini une sémantique pour le cas où les variables possèdent des valeurs entières. Nous avons proposé une extension au modèle de René Thomas pour des gènes qui assument à la fois un rôle d'activateur et d'inhibiteur [BCC+03]

6.3 Validation et Inférence de réseaux

L'objet de la *validation* de réseau est d'étudier des propriétés sur la représentation de la dynamique du réseau. Pour les réseaux génétiques, le modélisateur définit le réseau et des paramètres K, que l'on confrontera aux propriétés observées pour tenter de confirmer ou d'infirmer le modèle.

Les propriétés à valider sont écrites en formules logiques qui s'appliquent sur les traces du réseau. Chaque trace représente un chemin (ou une abstraction de celui-ci) dans le graphe d'états. La logique temporelle CTL (Computation Tree Logic) offre un langage formel pour exprimer des propriétés à propos de l'ensemble des traces d'un graphe d'états. Il est possible de valider algorithmiquement une formule par des techniques de Model Checking.

Cette approche est développée dans l'atelier *observabilité* de Génopole. Cet atelier cherche à proposer des méthodes de modélisation à la croisée de l'informatique et de la biologie. Compte-



Dans les modèle de régulation de réseaux génétiques fondé sur les réseaux de neurones récurrents (B), le niveau d'expression d'un gène i à l'instant $t + dt$ se calcule à partir à partir de ceux à l'instant t (y_j) et du *poids de chaque connexion* $w_{i,j}$. Un activateur (j) implique que $w_{i,j} > 0$ tandis qu'un inhibiteur implique que $w_{i,j} < 0$. Enfin, $w_{i,j} = 0$ si il n'y a pas d'interaction.
L'effet combiné des gènes sur la régulation du gène cible i (g_i) correspond à une somme pondérée de leur concentration (y_j) . S'ajoute à cette somme une constante (b_i) qui peut être interprétée comme le niveau basal du gène, soit :

$$\sum_j w_{i,j} y_j + b_i$$

Cette somme est composée à une fonction d'activation θ permettant de définir le taux d'expression du gène cible (i). Cette fonction peut être en sigmoïde [WWS99, Voh01a, Voh01b] ou l'identité (fonction inexistante) [MM98].
Le taux d'expression est modulé par une constante multiplicative (k_i). La dégradation des protéines synthétisées peut aussi s'intégrer. Le facteur de dégradation est modélisé par une équation de cinétique chimique du premier ordre $(\lambda_i.z_i)$. Le modèle de contrôle du taux d'expression gène cible $(\frac{dz_i}{dt})$ s'exprime par l'équation différentielle suivante :

$$\frac{dz_i}{dt} = k_i.\theta(\sum_j w_{i,j} y_j + b_i) - \lambda_i.z_i$$

ou sous la forme d'une différence finie :

$$z_i(t+1) = k_i.\theta(\sum_j w_{i,j} y_j(t) + b_i) - \lambda_i.z_i(t)$$

avec $z_i(t), y_j(t)$ la valeur de la concentration des gène i, j à l'instant t.
Sur la figure, le graphe A représente un graphe d'interaction signé. Rappelons que le signe + indique un activation et le signe − une inhibition. Le schéma B représente l'architecture du réseau de neurones récurrents lui correspondant. Le schéma C décrit les poids de connexions $w_{i,j}$ organisée sous forme de matrice de poids. Cette matrice est compatible avec le graphe A.

FIG. 6.2 – Régulation génétique modélisé par un réseau de neurones

tenu des conditions expérimentales en biologie d'une part et de la complexité des comportements à modéliser d'autre part, l'analyse des données de la biologie conduit à envisager cette approche pour faciliter la validation expérimentale des modèles proposés [PC03, BB03, GBC03].

Héritée des techniques du génie logiciel, l'emploi de CTL en biologie est tout à fait original. Il permet de valider les étapes d'un protocole d'expériences et de contribuer à sa définition en testant les hypothèses et les conclusions attendues sur la modélisation informatique. L'utilisation de CTL plutôt que d'une autre logique est principalement motivée par la disponibilité de l'outil de model-checking SMV [CAP86, CGL94, Che] considéré comme un standard dans ce domaine.

Le complément naturel de cette approche consiste à *inférer* le ou une partie du réseau à partir de propriétés ou de traces que le réseau doit vérifier ou engendrer. Il s'agit d'une approche complémentaire qui peut utiliser les mêmes outils. D'une manière pratique, on s'intéresse alors à définir un réseau et ses paramètres K pour produire un comportement spécifique, qui se définit soit par des propriétés que valide le réseau, soit par des traces que doit produire la simulation déduite de ce réseau. Ce comportement ne correspond pas nécessairement à un réseau unique mais à une classe de réseaux vérifiant les propriétés du comportement.

L'inférence peut se restreindre à certains paramètres d'un réseau. En particulier, il est possible de définir automatiquement les paramètres K, une fois la topologie établie. La méthode la plus élémentaire consiste à explorer toutes les valeurs possibles des paramètres K qui sont en nombre finis et de ne retenir que les valeurs répondant aux propriétés observées ou souhaitées. Pour des réseaux de taille modeste, la combinatoire qu'elle engendre reste maîtrisée.

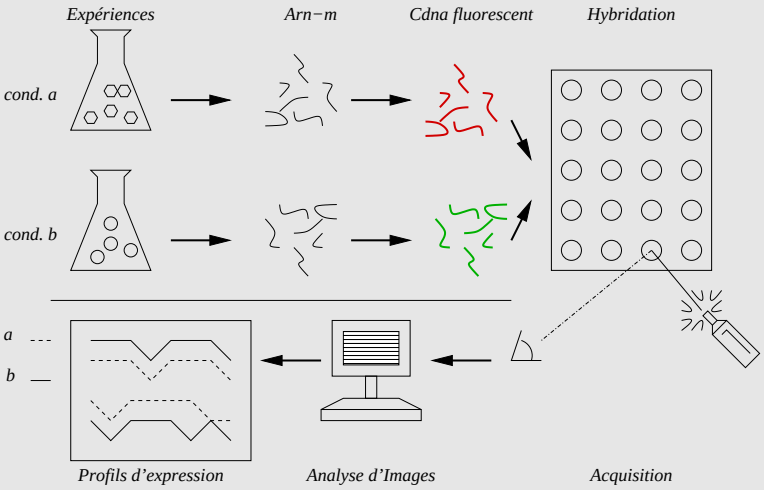
Le passage à la conception complète de réseaux *ab-initio* se heurte à une combinatoire très importante (le nombre de graphes orientés à n sommets est de 2^{n^2}). Elle conduit à limiter l'espace des paramètres entrant dans la définition des réseaux afin de réduire l'espace de recherche. Pour l'essentiel, la découverte des réseaux génétiques par une inférence automatique à partir des profils d'expression que produisent les puces à ADN (cf. figure 6.3) s'applique aux réseaux booléens [RS00]. Pour des raisons liées à l'explosion combinatoire, les réseaux inférés se révèlent alors insuffisamment précis par rapport aux modèles souhaités. Une solution naturelle se trouve alors dans une structuration modulaire du réseau.

6.4 Modularité des réseaux de régulation

La validation de modèles et l'inférence de réseaux correspondent à deux approches complémentaires qui examinent les relations entre modèles et propriétés de la dynamique des réseaux. Ces approches doivent se confronter au passage à l'échelle, car celui-ci est d'actualité. Pour l'organisme *Saccharomyces cervisiae* (la levure) on peut par exemple mentionner : un réseau de 490 gènes [GBBK02] signés provenant d'une étude bibliographique et [LRRa02] pour un réseau de plus de 2000 gènes orienté mais non signé extrait d'une analyse systématique sur les facteurs de transcription. Précisons que la levure possède approximativement 6000 gènes au total.

Si un consensus commence à s'établir sur la nature modulaire des réseaux de régulation [DGZ01, IBS+02, TR99, SSR+03], les méthodes permettant de distinguer les modules demeurent ouvertes. Nous nous proposons d'illustrer les orientations. Ces orientations se définissent en fonction des liens qui unissent, en biologie, la structure et la fonction. Il apparaît impossible de donner une définition brève et complète de ces termes tant la littérature est vaste sur ce sujet. Nous renvoyons le lecteur à l'article suivant [PC] pour une analyse plus approfondie sur ces liens. Succinctement, on peut définir une fonction d'un objet biologique (molécule, organes,...) comme le rôle qu'on lui attribue dans un

Les puces à ADN (ou biochips, biopuces, microarray) permettent l'étude de changement d'expression des transcrits (ARN) en considérant un très grand nombre de gènes sous l'influence des conditions étudiées. La technique des bio-puces se fonde sur l'hybridation des acides nucléiques (appariement des bases en A-T et G-C par des liaisons hydrogènes).
Une puce se constitue de centaines ou de milliers de *spots* qui correspondent à des fragments d'ADN fixés sur un support solide, une plaque de verre ou une membrane de nylon. Chaque séquence fixée est choisie pour caractériser un gène particulier.



L'expression des gènes dans des conditions environnementales fixées par l'expérience se mesure à partir de séquences d'ARN transcrites. On synthétise alors par une technique de transcription inverse les séquences d'ADN complémentaires correspondant aux séquences d'ARN-messagers transcrites. Les séquences d'ADN sont marquées par fluorescence et déposées sur la puce. Elles s'hybrident avec celles fixées sur la puce composant les spots. L'agglomération de séquences fluorescentes hybridées sur un spot permet de'apprécier la variation du niveau d'expression d'un gène.
Plutôt que de définir une mesure absolue d'expression de l'ARN messager, les résultats produits par les puces doivent être étudiés comme des indicateurs de changement d'expression. L'analyse de l'expression des transcrits s'effectue par comparaison entre deux conditions d'expériences. Ce protocole est du à l'absence de connaissances sur l'expression de référence d'un gène et à la variabilité des résultats d'expression obtenus qui rend la reproductibilité difficile à maîtriser.
Pour se faire, une technique consiste à réaliser une hybridation compétitive de la plaque entre deux lots de transcrits. Le premier correspond à celui d'une population de référence ; le second, à la population testée. Ces lots sont marqués de couleurs différentes par fluorescence qui permet leur distinction lors des traitements suivant l'hybridation. Les couleurs de marqueurs de fluorescences usuelles sont vert et rouge. Ils sont ensuite mélangés avant d'être déposés sur la plaque.
Par des traitements informatique d'analyse d'images faite par microscopie des spots illuminés par laser, on relève l'intensité des couleurs pour chacun des spots. Ce relevé détermine le ratio des deux intensités (*rouge/vert* généralement). La mesure reportée se définit donc relativement aux deux lots d'ARN.
Les différents ratios collectés pour une succession d'expériences seront rangés dans une *matrice de profil d'expression* où chaque ligne correspond à un gène et chaque colonne à une expérience. Les résultats se présentent aussi par *des courbes de profil d'expression* pour chaque gène où la valeur en ordonnée représente le ratio et celle abscisse le numéro d'expérience.
A partir de cette matrice, on identifiera les relations entre les différentes expressions des gènes. Les résultats des puces à ADN s'appliquent par exemple à la découverte :

- de profils caractéristiques d'expression correspondant à l'expression des gènes en réaction aux conditions d'expériences,
- de groupes de gènes *co-exprimés* dépendant d'un même système de régulation et dont les ratios évoluent de manière identique (ou de manière inverse),
- de dépendances causales entre gènes pour en déduire des réseaux génétiques.

FIG. 6.3 – Technologie des puces à ADN

environnement donné. La notion de fonction dépend donc de l'observation et de l'observateur. La structure d'un objet biologique représente les éléments constitutifs et les relations entre ces éléments qui sont le support de la fonction assurée par cet objet. Sans ces éléments, la fonction n'est plus assurée. Et, au niveau de l'analyse que l'observateur s'est fixée, on ne peut distinguer de la structure une autre fonction. La définition de ces termes demeurent très générale à ce niveau. Cependant, comme nous le verrons, elle devient plus précise une fois les orientations introduites. Les deux approches pour définir des modules sont : une approche orientée structure et une approche orientée fonction.

Approche orientée structure L'approche *orientée structure* détermine les modules sur la base de la structure topologique du réseau. Un module correspond à une sous-partie connexe du graphe répondant à une propriété topologique particulière (cycle, chemin, arbre...) et définie par le modélisateur. Idéalement, cette propriété est caractéristique d'une fonction spécifique. Une approche, initiée conjointement par U. Alon [SSOA02, MSOI+02] et N. Guelzim, S. Bottani, P. Bourgine et F. Képès [GBBK02], vise à inventorier toutes structures pouvant présenter des propriétés particulières. Sans interpréter le rôle des structures, elle repère toutes structures sur-représentées. Ces structures se nomment des *motifs de graphe*. Selon ces auteurs, un motif de graphe est un graphe particulier. L'étude fut menée pour des petits graphes à trois sommets. Bien que différente dans la méthode de recherche, cette analyse a été déjà précédemment utilisée par René Thomas [TSRT95] pour l'identification de cycles dans les réseaux qui correspondent à des motifs dans la mesure où ils assurent un contrôle particulier.

Nous dégageons dans cette approche deux phases :

- *La première consiste à inventorier des motifs caractéristiques.* Cet inventaire dépend des critères d'extraction des motifs. La sur-représentation est le critère choisi par U. Alon. Ce critère présente les limites suivantes : la première tient de l'hypothèse d'une corrélation entre la sur-représentation et l'importance du motif. Cette hypothèse peut exclure des motifs essentiels sous-représentés. La seconde concerne la combinatoire des sous-réseaux explorés qui limite pratiquement l'exploration à de petits motifs possédant un nombre de sommets restreints.
- *La seconde recherche des instances de motifs caractéristiques inventoriés à la phase précédente.* Cette phase a pour objet de structurer le réseau par des instances de motifs connus. Elle renseignerait sur la nature régulatrice de celui-ci car elle fournirait un découpage de sa topologie selon des instances de motifs caractéristiques auxquels s'associeraient un contrôle particulier. Algorithmiquement, la recherche de motifs dans un réseau s'assimile à la recherche de sous-graphes dans un graphe donné. Il correspond au problème d'isomorphisme de sous-graphes qui est un problème NP-complet [Ata99].

L'approche orientée structure cherche d'abord des motifs caractéristiques afin d'en proposer une interprétation. Cette phase est plus délicate car il faut s'assurer de la pertinence du motif qui repose sur deux questions Est-il élémentaire ? Est-il utilisé dans le réseau de la manière dont il s'interprète de manière isolée ? Cette association s'appuie sur une analyse qui établirait *la modularité du contrôle exercé*. Elle se fonde soit sur la sémantique associée au réseau si l'on raisonne sur une modélisation, soit sur une étude expérimentale qui est l'objet de l'approche suivante.

Approche orientée fonction (biologique) Une approche duale à la précédente consiste à déduire les structures après avoir identifiées le contrôle exercé. Schématiquement elle propose de remonter de la fonction à la structure, à l'inverse de l'approche précédente.

En 2001 E. Davidson [CHBBD01, Da02] rapporte les conclusions d'une analyse de 30 année sur

l'oursin de mer. Par une analyse quantitative, son équipe a été à même de définir complètement les unités fonctionnelles et la relation des gènes qui contrôlent le développement de l'endoderme et du mésoderme chez l'oursin de mer. Le réseau a été défini à partir d'un grand nombre de perturbations dans lequel les gènes sont altérés. Cette étude met en évidence l'existence de structures modulaires du réseau génétique pour ce processus. La structuration du réseau en module arrive *a posteriori* d'une analyse biologique qui guide sa construction.

La sous-section 6.4.1 décrit une méthode que nous avons proposée pour étendre l'approche orientée structure afin de définir des modules. Elle s'inspire d'une technique d'analyse statique utilisée en compilation.

6.4.1 Contribution à la définition de Module

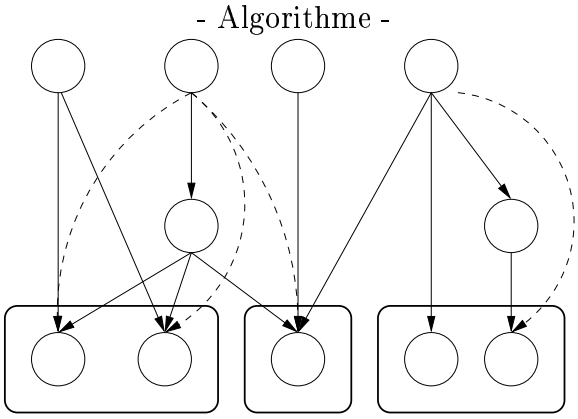
Nous avons proposé une «expérience numérique» [Del02] qui permet d'avancer dans la définition de modules. Ces modules seront nommés $\sigma-$modules afin de souligner leur origine algorithmique. Une des questions centrale relative à leur calcul est de valider leur pertinence par rapport aux fonctions biologiques. Lorsqu'un module présente une définition se rapportant à une fonction, nous dirons qu'il est *fonctionnellement cohérent* (ou cohérent). Cette cohérence est *a priori* difficile à apprécier car elle dépend des fonctions biologiques. Cependant, en s'appuyant sur les bases de données biologiques, il est possible de proposer un calcul de de score qui estime cette notion. Pour compléter la validation, nous confronterons ce score à une construction aléatoire des modules. L'algorithme de calcul de σ-Modules (σ-MC) a été testé sur le réseau de 490 gènes de la Levure précédemment mentionné.

Dans notre cas, la cohérence fonctionnelle se calcule grâce à la classification MIPS [MIP] qui classifie les gènes par rapport aux fonctions biologiques de la protéine qu'ils codent. La classification MIPS organise la définition des fonctions biologiques en catégorie selon une structuration hiérarchique. Par exemple, la catégorie 3 correspond au cycle cellulaire et aux traitements relatifs à l'ADN. La catégorie 3.3 concerne le cycle cellulaire ; la catégorie 3.3.2 la méiose (division cellulaire sexuée). Plusieurs classes fonctionnelles sont attribuées à chaque gène. Les gènes des $\sigma-$modules que nous avons calculés se répartissent dans 72 classes fonctionnelles différentes. Un $\sigma-$module est cohérent si tous ses gènes possèdent au moins une classe MIPS commune. Le score de cohérence correspond au pourcentage de modules cohérents construits par σ-MC.

Intuitivement, un $\sigma-$module désigne *un ensemble de gènes corrégulés par un ensemble identique de régulateurs*. Des travaux plus récents de calculs de modules à partir de données de puces à ADN adoptent une définition similaire pour caractériser des «Networks-modules» [SSR+03].

L'algorithme de σ-MC est le suivant : nous considérons des gènes «d'entrée» qui ne possèdent pas de prédécesseurs dans le réseau à l'exception d'eux mêmes. De même, nous considérons des gènes de «sortie» qui ne possèdent pas de successeurs dans le réseau à l'exception d'eux même. σ-MC définit une partition des gènes de sortie selon la propriété suivante : les gènes de sortie d'un même $\sigma-$module sont reliés de manière directe ou indirecte (par transitivité) aux même gènes d'entrée (cf. figure 6.4). *Les $\sigma-$modules seront assimilés à des groupes de gènes de sortie.*

83% des $\sigma-$modules construits par σ-MC sont cohérents. En comparaison, une construction aléatoire de modules fournit un résultat inférieur à 1% (cf. figure 6.4). Il faut cependant noter que les $\sigma-$modules ne possédant qu'un gène sont trivialement cohérents. En restreignant cette évaluation à ceux ayant au moins deux gènes, 58.2% d'entre eux demeurent cohérents. Ces résultats indiquent clairement une relation entre les $\sigma-$modules calculés *a priori* à partir du réseau et la classification

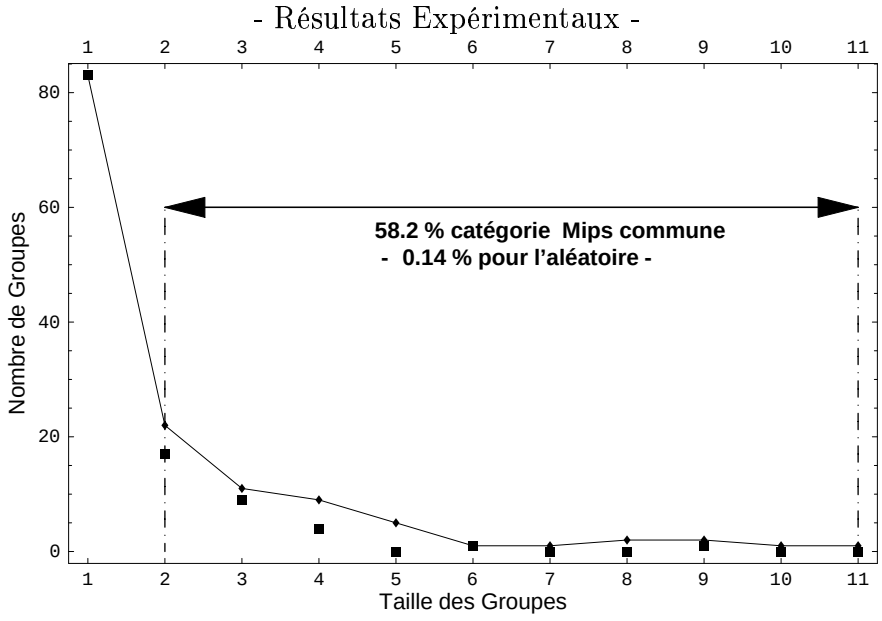


L'algorithme de construction des groupes est le suivant : Soit $R = (V, E)$ un réseau génétique orienté :

1. Calculer la fermeture transitive du réseau $Cl(R) = (V, Cl(E))$
2. Identifier les sommets sans prédécesseur à l'exception d'eux mêmes. Nous appellerons cet ensemble $\Delta_0^- \subseteq V$ Ces sommets correspondent aux gènes d'entrée.
3. Identifier les sommets sans successeur à l'exception d'eux mêmes. Nous appellerons cet ensemble $\Delta_0^+ \subseteq V$ Ces sommets correspondent aux gènes de sortie.
4. Former des groupes de gènes de Δ_0^+ ayant exactement les même prédécesseurs dans Δ_0^-, soit $\mathcal{P} = \{\gamma \subseteq \Delta_0^+\}$, la partition des sommets vérifiant cette propriété, soit $\delta^-_{Cl(R)} : V \mapsto \mathbf{2}^{Cl(E)}$ l'ensemble des prédécesseur d'un sommet on a :

$$\forall \gamma \in \mathcal{P}, \forall v_1 \in \gamma, \forall v_2 \in \gamma \ : \delta^-_{Cl(R)}(v_1) \cap \Delta_0^- = \delta^-_{Cl(R)}(v_2) \cap \Delta_0^-$$

La figure décrit un exemple de regroupements formés à partir de l'algorithme. les lignes en pointillé correspondent à celles ajoutées lors du calcul de la fermeture.
La construction aléatoire des groupes s'effectue ainsi : on attribue à chaque gène de sortie un nombre variant de 1 au cardinal de l'ensemble des gènes d'entrée. Les gènes de sortie font partie de la même partition si ils possèdent le même nombre généré aléatoirement.



Les courbes correspondent respectivement à :
- pour la courbe en trait plein, le nombre de gènes ventilés selon la taille des groupes trouvées ;
- pour la courbe avec des carrés, le nombre de gènes ventilés par groupe possédant au moins une catégorie fonctionnelle commune.

FIG. 6.4 – Structures modulaires

fonctionnelle MIPS.

Transposé dans un contexte informatique, σ-MC indique la présence de blocs de programmes pouvant être encapsulés dans une procédure ou une fonction. En effet, en modélisant un programme par un graphe de flots de données, les sommets représentant les paramètres d'entrée d'une procédure sont connectés de manière directe ou indirecte à ceux représentant les variables de sortie de cette procédure. Cette notion a inspiré la définition et l'algorithme de calcul des σ-modules.

D'une manière immédiate, σ-MC peut servir à vérifier la construction des réseaux en indiquant les $\sigma-$modules incohérents. Ces derniers révéleraient des erreurs putatives dans la construction du réseau. σ-MC a déjà permis d'identifier, dans le graphe analysé, une anomalie dans un $\sigma-$module de la méiose qui révélerait une erreur dans une connexion. L'origine de cette erreur proviendrait d'une homonymie entre des noms de gènes ; erreur difficile à identifier car il est nécessaire de chercher à nouveau les données originales dans la bibliographie.

6.4.2 Résumé des approches

Nous avons abordé la définition de modules par des méthodes algorithmiques et leur validation par rapport aux fonctions biologiques (sous section 6.4.1). Dans cette partie rassemblant des idées prospectives, nous ramènerons ces éléments à la modélisation discrète de la dynamique des réseaux décrites à la section précédente (section 6.2). Nous avons introduit deux approches d'analyse de la modularité : la première identifie les modules par une analyse topologique du réseau, la seconde par un regroupement de gènes corrégulés. Si l'on rapproche celles-ci de l'expression des gènes on peut distinguer deux objets différents pour l'étude de la modularité : les *motifs* et les *modules*.

Les motifs caractérisent des *opérateurs génériques*. Leur découverte peut se faire à partir d'une description statique du réseau car leur généricité impose que les lois régissant leur dynamique soient invariantes. Il serait possible d'inférer la dynamique de la topologie. L'analogie à l'informatique serait celle des circuits.

Les modules caractérisent des *groupes de gènes corégulés coopérant à la réalisation d'une fonction* en réponse à la régulation exercée. Leur analyse dépend à la fois de la dynamique et des fonctions observées. L'observation de ces fonctions conduit à organiser le réseau en module assurant une fonction donnée. Cependant, pour rendre compte de la dynamique du système en regard des fonctions qu'il réalise, il apparaîtrait nécessaire de faire évoluer cette organisation. En effet, les gènes apparaissant associés à différentes fonctions, l'organisation en module devrait aussi évoluer en fonction de l'expression des gènes. L'appartenance d'un gène à plusieurs classes fonctionnelles MIPS va dans ce sens.

De récents travaux sur la définition de modules [SSR+03] remarquent aussi l'existence d'une organisation de la régulation où des modules sont caractérisés *«en partie par des recoupements de combinaisons de motifs et de régulateurs* [3]*»* .

L'absence de partition des réseaux de régulation génétiques et la dynamique des constructions modulaires conduisent à une analyse en partie différente de celle abordée pour la modularité des systèmes informatiques. A la dynamique du système s'ajouterait celle d'une dynamique d'organisation en module. Actuellement, nous ne voyons pas d'analogie avec des systèmes informatiques qui permette de rendre une image totalement correcte de cette organisation biologique.

[3] "On a global scale, it also suggests a higher organization of combinatorial regulation [..] in which distinct modules are characterized by partly overlapping combinations of *cis*-regulatory motifs and regulators.".Précisons que le terme de motifs se rapportent à un signal consensus spécifique du promoteur.

6.5 Bilan des travaux en cours

La modélisation des réseaux génétique a pour objet de comprendre la régulation exercée par les gènes. Elle s'appuie sur l'étude de sa dynamique. Face à une dynamique complexe, la simulation dans un cadre approprié constitue une approche possible . Nous avons développé un simulateur pour un réseau de régulation [GMD01, JLGD03].

Une alternative consiste à définir des outils d'analyse de la dynamique d'un système complexe. A la fois grâce à une analogie possible entre gènes et processus concurrents et parce que cette analyse se destine à être traitée par ordinateur, son étude utilise des méthodes formelles employées en informatique. Elle se fonde en premier sur la définition d'une sémantique qui détermine les règles de calculs d'un graphe d'états à partir du réseau de régulation. Nous avons proposé une sémantique formalisant la modélisation de René Thomas qui étend la possibilité de définir à la fois une régulation positive et négative pour une même connexion. La sémantique a été faite en collaboration avec l'équipe vérification des systèmes embarqués de l'IRCYN [BCC+03] et C. Müller, en stage de DEA (AMIB), co-encadré avec J.P. Comet, a développé un logiciel d'analyse de réseau se fondant sur cette sémantique.

La définition de la sémantique correspond à la première étape d'analyse des systèmes biologiques par des méthodes de model checking. L'emploi de CTL en biologie est tout à fait original. Il s'agit de s'appuyer sur ses techniques afin d'assister l'expérimentation en caractérisant formellement les étapes du protocole. Cette approche est développée par l'atelier Observabilité de génopole. Actuellement son application se destine à des réseaux possédant un nombre restreint de gènes.

Le passage à l'échelle des modèles et des méthodes évoqués place la modularité comme une pierre angulaire de cet édifice. Elle est nécessaire autant pour aborder la complexité des systèmes biologiques dans une perspective d'intégration que pour définir une stratégie efficace de traitement fondée sur son incrémentalité. Elle pose une question ouverte sur la définition de modules pour laquelle nous avons proposée une contribution. Nous avons proposé une approche inspirée de méthodes d'analyses statiques employée en compilation pour structurer des programmes. La définition de structures modulaires permet d'envisager de modéliser des systèmes biologiques de plus en plus complexes et intégrés.

Pour étendre cette approche, nous souhaiterions introduire des notions d'analyse de modules empruntées à l'analyse statique, l'interprétation abstraite et à la sémantique des langages de programmation. Schématiquement, nous envisagerions d'étudier ce réseau à la fois comme un langage et comme un réseau à flots de données. Dans le premier cas, la syntaxe correspond aux motifs et la sémantique aux fonctions biologiques. La seconde assimilation permettrait d'étudier des notions de compositionalité, d'observabilité et de cohérence des stuctures modulaires correspondant aux sous-réseaux.

A terme, la simulation ou l'analyse d'une cellule complète est visée. Le groupe de réflexion CELLÏA [Cel] dont j'étais le coordinateur (2000-2002), avait pour but d'examiner les méthodes permettant cette intégration. Il fut un des groupes précurseurs des ateliers Génopole, EPIGÉNÈSE [Epi] qui reprend cet objectif lointain en lui fixant des étapes intermédiaires constituant les thématiques des ateliers.

Curriculum Vitae

Cursus

Oct. 2000 -	Thématique bio-informatique	U.E.V.E LAMI UMR 8042, GENOPOLE, équipe BIOINFO
Oct. 1994 - 2000	Maître de conférences	U.E.V.E LAMI UMR 8042, équipe PARALL - OPAL
Oct. 1993- Oct. 1994	ATER	Paris XI, LRI, équipe Architecture,
1991-1993	Thèse (MESR-Moniteur)	Paris XI, LRI, équipe Architecture,

Titulaire d'une Prime d'Encadrement Doctorale depuis 1998 (renouvelée en 2002)

Recherches

Thématiques

Bio-Informatique	
TAGCC Outils d'annotation et de prédictions des gènes	2000 -
Analyse des réseaux de régulation	2000 -
Parallélisme irrégulier	
PARADEIS, Composants logiciels pour le parallélisme et l'irrégularité	1995 - 2000
Parallélisation automatique des programmes manipulant des matrices creuses	1995 - 2000
Compilation des communications	
Compilation des communications pour HPF	1995-1997
(Thèse) Compilation des communication pour un reséau statique	1992-1995

Participations aux actions scientifiques

Membre du comité de programmes de Renpar	2003
Action IMPG Simulation de processus biologiques dans le contexte de la génomique	2001 -
Coordinateur du groupe CELLÏA	2000-2001
GDR AMN , Thème IHPERF	1996 - 2000
Action PRO Projet inter-GDR ARP, ALP	1998 - 2000
Action PARADIGME, GDR ALP	1992 - 1994

Encadrements de 3^{me} cycle

Nom	DEA/Thèse	Sujet	Année	% d'encadrements
R.Adle	DEA + Thèse	Outil de parallélisation automatique des programmes denses pour des structures creuses	1995, 1996 - 1999	100%, 90%
D. Remy	DEA + Thèse	PARADEIS, un environnement de développement parallèle pour le traitement du creux	1996, 1997 - 2000	100%, 90%
E.Fayolle	DEA	Analyse et comparaison de langages pour le parallélisme	1999	100%
L. Rival	DEA	Identification de critères efficaces pour l'annotation *ab-initio* en TAGCC	2001	100%
A. Dubas	DEA	Mécanisme de résolution en TAGCC	2002	100%
S. Djebali	DEA + Thèse	Outils de génération automatique de programmes en TAGCC	2002	100%, 50%
M. Manceny	DEA	Annotation par mécanisme de plus court chemin	2003	100%
I. Martinat	DEA	Langage de description de modèles génétiques	2003	50%
C. Müller	DEA	Sémantique des réseaux de régulation	2003	50%

Réalisations Logicielles

Gene Network Functional Browser, en cours	2001 -
Compilateur TAGCC v 1.3, logiciel libre, $\approx$ 7000 lignes	2000 -
Plateforme PARADEIS, logiciel libre, $\approx$ 10000 lignes	1998 - 2000
Maquette de compilateur de communications, $\approx$ 1000 lignes	1997
HPT $\approx$ 5000 lignes	1993 - 1995

Enseignements

Depuis ma nomination sur un poste de maître de conférences à Evry, j'ai mis en place 5 enseignements différents en premier second et troisième cycles.

1^{er} cycle	Algorihtmique, Langage C
2^{eme} cycle	Intelligence artificielle, Parallélisme, Réseau, Architecture des ordinateurs
3^{eme} cycle	Compilation parallèle, Programmation en Bioinformatique (AMIB)

J'ai été le responsable de la définition du cursus informatique pour une filière de Bio-Informatique créée en 1998 - 1999.

Activités Administratives

Co-responsable du DEA AMIB	2003 -
Responsable du Deug MIAS 2	2000 - 2003
Membre de la commission de spécialiste de la section 27	1995 -
Président B de commission mixte des sections 27-65	2000,2001
Responsable pour la partie informatique de l'IUP Génie Biologique et Informatique	1998 - 2000
Membre du Conseil d'Administration de l'université	1997 - 1998

Publications

Ouvrages

G. BERNOT, J.MICHEL-GUESPIN, P. AMAR, A.ZERMILINE, J.P. COMET, F. DELAPLACE, P. BALLET,*Modélisation, observabilité et Expérimentation :Etudes d'un cas.*
Modélisation et simulation de processus biologiques dans le contexte de la génomique, ISBN2-84704-036-6

Revues internationales avec comité de rédaction

R. ADLE, M. AIGUIER, F. DELAPLACE,*Automatic parallelization of sparse matrix computation : a static analysis.*
à paraître, Journal of Parallel and Distributed Computing, (34 pages)

J.L. GIAVITTO, O. MICHEL, F. DELAPLACE,*Declarative Simulation of Dynamical systems : the 8,5 programming language and its application to the simulation of genetic networks.*
Biosystem, Vol. 68, 2-3, Fevrier-Mars 2003, Pages 155-170

F. KEPÈS, F. DELAPLACE, J.M. DELOSME, J. GUESPIN, R. INCITTI, V. NORRIS,*Modeling and simulating biological processes in the genomic era : an account of a multidisciplinary seminar held in Autrans.*
Journal of Biological Physics and Chemistry 2 (2002), pages 62–66

P. Amar, P. Ballet, G. Barlovatz-Meimon, A. Benecke, G. Bernot, Y. Bouligand, P. Bourguine, F. Delaplace, J.M. Delosme, M. Demarty, I. Fishov, J. Fourmentin-Guilbert, J. Fralick, J.-L. Giavitto, B. Gleyse, C. Godin, R. Incitti, F. Képès, C. Lange, L. Le Sceller, C. Loutellier, O. Michel, F. Molina, C. Monnier, R. Natowicz, V.Norris, N. Orange, H. Pollard, D. Raine, C. Ripoll, J. Rouviere-Yaniv, M. Saier jnr., P. Soler, P. Tambourin, M. Thellier, Ph. Tracqui, D. Ussery, J.-C. Vincent, J.-P. Vannier, P. Wiggins, A. Zemirline, *Hyperstructures, Genome Analysis and I-Cells.*
Acta Biotheoretica, 50 :357-373, 2002

C.Germain, F.Delaplace, R.Carlier,*A static execution model for data parallelism.*
Parallel Processing Letter, 4 :367–378, Decembre 1994

Revues nationales avec comité de rédaction

D. Remy, F. Delaplace,paradeis *: une extension de STL pour le calcul parallèle creux.*
Techniques et Sciences Informatiques, 19(9),Novembre 2000

R.Adle, F. Delaplace,*Vers la parallélisation automatique des programmes manipulant des matrices creuses.*
Techniques et Sciences Informatiques, 18(7) :715-744, Juillet 1999

F.Delaplace, F.Cappello,*Compilation des communications dans les langages data parallèles.*
Techniques et Sciences Informatiques, 12(6) :775–797, Decembre 1993

Conférences internationales avec comité de sélection

J.L. Giavitto, O. Michel, F. Delaplace,*Declarative Simulation of Dynamical systems : the 8,5 programming language and its application to the simulation of genetic networks.*
IPCAT 2001, 2001

R. Adle, M. Aiguier, F. Delaplace,*Automatic parallelization of sparse matrix computation : a static analysis.*
EUROPAR 2000, Septembre 2000

F. Delaplace and D. Rémy,paradeis *: An STL Extension for Data Parallel Sparse Matrix Computation.*
Vecpar 2000, Juin 2000

F. Delaplace and D. Rémy,paradeis *: An object library for parallel sparse array computation.*
APC99, Salzburg,Austria, Fevrier 1999.

F.Delaplace, R.Adle,*Extension of the dependance analysis for the sparse computation .*
PDCS'97

C.Germain, F.Delaplace,*Compiling Block Cyclic Distribution.*
PARCO 1997

C.Germain,F.Delaplace,*Automatic vectorization of communications for data-parallel programs.*
Euro-par 95, number 966 in LNCS. Springer-Verlag, Aout 1995

F.Cappello, J.L.Bechennec, F.Delaplace, C.Germain, JL.Giavitto, V.Neri, D.Etiemble, *A parallel architectures based in compiled schemes.*
PARCO, 1993

F.Cappello, J.L. Bechennec, F.Delaplace, C.Germain, J.L. Giavitto, V.Neri, D.Etiemble, *Balanced distributed memory for parallel computers.*
International conferences on parallel processing, volume 1, pages 72–76, St Charles, USA, August 1993. CRC PRESS INC

F.Delaplace,F.Cappello,*Data layout impact on communications compilation for synchronous msimd machine.*
Euromicro, pages 469–476, Paris, France, 1992

F.Delaplace,J.L.Giavitto,*An efficient routing strategy for process migration.*
Euromicro, pages 153–160, Vienna, Austria, 1991

Conférences nationales avec comité de sélection

R.Adle, F.Delaplace,*Vers l'automatisation de la parallélisation des traitements creux.*
Renpar'9, Lausanne, Suisse, mai 1997

F.Cappello, JL.Giavitto, JL.Bechennec, F.Delaplace,*Architectures parallèles équilibrées.*
Congrès AFCET, Versailles, France, 1993

F.Delaplace, C.Germain,*Test d'identification des communications statiques.*
Renpar 5, pages 65–68, Brest, France, May 1993

F.Delaplace, F.Cappello,*Compilation pour une machine MSIMD synchrone.*
Renpar 4, Lille, France, 1992

Atelier avec comité de sélection , école

G. Bernot, O. Roux, C. Auberger, V. Bassano, F. Cassez, J.-P. Comet, F. Delaplace, A. Richard, O. Roux, F. Tahi,*Temporal logics and biological regulatory networks.*
Modeling and Simulation of Biological Regulatory Processes, ECCB Satellite Meeting, Octobre 2003

G. Bernot, F. Cassez, J.P. Comet, F.Delaplace, C. Muller, O. Roux, O. Roux,*Semantics of Biological Regulatory Networks.*
BioConcur 2003, ENCS, Marseille

G. Bernot, J.Michel-Guespin, A.Zermiline, J.P. Comet, F. Delaplace, P. Ballet, P. Amar,*Modélisation, observabilité et Expérimentation :Etudes d'un cas.*
Modélisation et simulation de processus biologiques dans le contexte de la génomique, Autrans, Mars 2002

R.Adle, F.Delaplace,*Vers la parallélisation automatique des programmes manipulant des matrices creuses.*
workshop Compilation et Parallelisation Automatique, Obernay, Octobre 1999, Invité

D. Remy , F. Delaplace,*Eléments de conception d'une STL data-parallèle pour le traitement des matrices creuses .*
3ieme séminaire sur l'algorithmique numérique appliquée aux problèmes industriels, Mars 1999

R.ADLE, F.DELAPLACE,*analyse de dépendances pour des programmes manipulant des matrices creuses.*
CoCa'98,Invité

R.ADLE, F.DELAPLACE, *Extension de l'analyse de dépendances pour des programmes manipulant des matrices creuses.*
Ecole ICARE 1997, Invité

Posters-Communications orales

S. DJEBALI, F.DELAPLACE, H.R. CROLLIUS,*Exogean : an Expert on Eukaryotic Gene Annotation.*
Poster ECCB, Septembre 2003

F. DELAPLACE, TAGCC *: un langage dédié à l'annotation des séquences.*
Réunion du 20-21 janvier 2003 Loria, Nancy

F.DELAPLACE, *Toward a static analysis for the study of properties of gene networks.*
Poster Symposium on macro molecular networks, Juillet 2002

F. DELAPLACE, S. DJEBALI,TAGCC *an equational language dedicated to biosequence analysis.*
Poster JOBIM, 2002

Thèse & Rapports techniques

F. DELAPLACE,*Gatcc, a language based on Transducers and Constraints dedicated to Biosequences Analysis.*
Rapport Technique 66,LaMI, Septembre 2001

D. REMY, F. DELAPLACE, PARADEIS *: An STL Extension for Data Parallel Sparse Matrix Computation.*
Rapport Technique 41, LaMI, Avril 1999

R. ADLE, M. AIGUIER, F. DELAPLACE,*Automatic parallelization of sparse matrix computation : a static analysis.*
Rapport Technique 42, LaMI, Décembre 1999

F.DELAPLACE,R.ADLE,*Extension of dependance analysis for the sparse computation.*
Rapport Technique 22, LaMI, Avril 1997

F.DELAPLACE,*Parallelization for Sparse Matrix : an Automatic Approach.*
Rapport Technique 14, LaMI, Avril 1995

F.DELAPLACE,*Compilation des communications dans un langage data-parallèle pour les architectures à réseau à communications compilées.*
Thèse, Université de Paris XI, LRI, Fevrier 1994

C.GERMAIN, F.DELAPLACE, R.CARLIER,*A static execution model for data parallelism.*
Rapport Technique 862, Laboratoire de Recherches en Informatique, LRI,France, Septembre 1993

Soummission à des revues internationales

F. DELAPLACE,*Egdes Motifs Gammars and its Application to Biological Regulatory Networks.*
Soummis en Juillet 2003, Theoretical Computer Science

F. Delaplace, M. Manceny,*The Gene Game : application of the game theory to gene networks.*
Soummis CMSB 2004

Bibliographie

[AAD00] R. Adle, M. Aiguier, and F. Delaplace. Automatic parallelization of sparse matrix computations : A static analysis. In A. Bode and al, editors, *Euro-par 2000*, volume 1900, pages 340–348. LNCS, Springer Verlag, September 2000.

[AD99] R. Adle and F. Delaplace. Vers la parallélisation automatique des programmes manipulant des matrices creuses. *Technique des Sciences Informatiques*, 18(7) :715–744, July 1999.

[Adl99] R. Adle. *Outils de parallélisation automatique des programmes denses pour les structures denses.* PhD thesis, Université d'Evry Val d'Essonne, January 1999.

[Agg84] A. Aggarwal. *The art gallery theorem : its variations, applications and algorithmic aspects.* PhD thesis, Dept. of Electrical engineering and computer science, The John Hopkins University, 1984.

[Ana] Genetic Linkage Analysis. A bibliography on computational gene recognition. `http ://linkage.rockefeller.edu/wli/gene/`.

[Anc91] C. Ancourt. *Génération automatiques de codes de transfert pour multiprocesseurs à mémoire locales.* PhD thesis, Ecole Nationale Supérieure de Mines de Paris, March 1991.

[ASU86] A. V. Aho, R. Sethi, and J. D. Ullman. *Compilers : Principles, Techniques and Tools.* Addison-Wesley, Inc., Reading, Mass., 1986.

[Ata99] M. J. Atallah, editor. *Algorithms and Theory of Computation Handbook.* CRC Press, 1999. Subgraph isomorphism is NP-comple.

[Atl86] H. Atlan. *A tord et à raison.* Points Science, 1986.

[Aut94] J. M. Autebert. *Théorie des langages et des automates.* Masson, 1994.

[Ban88] U. Banerjee. *Dependence Analysis for Supercomputing.* Kluwer Academic Publisher, 1988.

[BB03] V. Bassano and G. Bernot. Marked regulatory graphs : A formal framework to simulate biological regulatory networks with simple automata. In *RSP-Workshop*, San Diego, June 2003.

[BBCR98] T. Brandes, F. Brégier, M. C. Counilh, and J. Roman. Contribution to better handling of irregular problems in HPF2. *LNCS*, 1470 :639–649, 1998.

[BCC+03] G. Bernot, F. Cassez, J.P. Comet, F. Delaplace, Müller C., O. Roux, and O. Roux. Semantics of biological regulatory networks. In *Submitted to Bio-Concur*, volume ?, page ?, September 2003.

[Ber66] A.J. Bernstein. Analysis for parallel processing. *IEEE Transactions on Electronic Computers*, EC-15(5), October 1966.

[BK97] C. Burge and S. Karlin. Prediction of complete gene structure in human genomic dna. *Journal of Molecular Biology*, 268 :78–94, 1997.

[Boi74] N. Boileau. *Art poétique.* -, 1674.

[Bou96] Luc Bouge. The data parallel programming model : A semantic perspective. In *The Data Parallel Programming Model*, pages 4–26, 1996.

[BW95] A.J.C Bik and H.A.G Wijshoff. Advanced compiler optimizations for sparse computations. *Journal of Parallel and Distributed Computing*, 31 :109–126, 1995.

[BW96] A.J.C Bik and H.A.G Wijshoff. Automatic data structure selection and transformation for sparse matrix computation. *IEEE Transactions on Parallel and Distributed Systems*, 7(2) :1–19, 1996.

[CAP86] E. M. Clarke, Emerson E. A., and Sistla A. P. Automatic verification of finite-state concurrent systems using temporal logic specification. *ACM Transactions on Programming Languagess and Systems*, 8(2) :244–263, 1986. SMV related article.

[Cap94] F. Cappello. *PTAH : étude d'une architecture parallèle à ressources équilibrées et communications compilées.* PhD thesis, Université de Paris XI, LRI, 1994.

[CD79] B. Chazelle and D. Dobkin. Decomposing a polygon into its convex parts. In *Proceedings of the 11th Annual ACM Symposium on Theory of Computing*, pages 38–48, 1979.

[CD02a] O. Cinquin and J. Demongeot. Positive and negative feedback : striking a balance between necessary antagonists. *Journal of Theoretical Biology*, 216(2) :229–41, 2002.

[CD02b] O. Cinquin and J. Demongeot. Roles of positive and negative feedback in biological system. *C. R. Biol*, 325(11) :1085–1095, 2002.

[Cel] Cellia. Cellia : Groupe de travail en simulation des processus biologiques. `http ://cellia.lami.univ-evry.fr`.

[CGL94] E. Clarke, O. Grumberg, and D. Long. Verification tools for finite-state concurrent systems. *LNCS*, 803, 1994. SMV related article.

[CGST93] S. Chaterjee, J.R Gilber, F.J.E Schreiber, and S.H. Teng. Generating local addresses and communications sets for data parallel programs. In 4^{th} *ACM Sigplan Symposium of Principles and Practices of Parallel Programming*, 1993.

[Cha80] B. Chazelle. *Computational Geometry and Convexity.* PhD thesis, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, July 1980.

[CHBBD01] Y. Chiou-Hwa, H. Bolouri, J. M. Bower, and E. H. Davidson. *Computational Modeling of Genetic and Biochemical Networks*, chapter A logical Model of Cis-Regulatory Control in a Eukariotic System. MIT press, 2001.

[Che] Model Checking. Model checking at cmu. `http ://www-2.cs.cmu.edu/ modelcheck/`.

[CMZ92] B. M. Chapman, P. Mehrotra, and H. P. Zima. Programming in Vienna Fortran. *Scientific Programming*, 1(1) :31–50, 1992.

[Coe96] F. Coelho. *Contribution à la compilation de High Performance Fortran.* PhD thesis, Ecole des Mines de Paris, 1996.

[CZM94] B. Chapman, H. Zima, and P. Mehrotra. Extending hpf for advanced data parallel applications. *IEEE Parallel & Distributed Technology*, 2(3) :59–70, 1994.

[Da02] E. Davidson and al. A genomic regulatory network for development. *Science*, 295(5560) :1669–16678, March 2002.

[DC03] V. Danos and Laneve C. Graphs for core molecular biology. In *CMSN'03*, 2003.

[DDC03] S. Djebali, F. Delaplace, and H. R. Crollius. EXOGEAN : an expert on eukaryotic gene annotation. Poster - Symposium on macro molecular networks, Sept. 2003.

[DEKM98] R. Durbin, S. R. Eddy, A. Krogh, and G. J. Mitchison. *Biological Sequence Analysis : Probabilistic Models of Proteins and Nucleic Acids.* Cambridge University Press, 1998.

[Del01] F. Delaplace. Gatcc a language based on transducers and constraints dedicated to biosequences analysis. Technical Report 66, LaMI, September 2001.

[Del02] F. Delaplace. Toward a static analysis for the study of properties of gene networks. Poster - Symposium on macro molecular networks, July 2002.

[DER86] I.S. Duff, A.M. Erisman, and J.K. Reid. *Direct Methods for Sparse Matrices.* Oxford Sciences Publications, 1986.

[DF02] F. Dardel and F.Kepes. *Bioinformatique. Génomique et post-génomique.* Ecole polytechnique, 2002.

[DGZ01] VR. Davuluri, I. Grosse, and MQ. Zhang. Computational identification of promoters and first exons in the human genome. *Nature Genetics*, 29 :412–417, 2001.

[d'H99] P. d'Haeseleer. Linear modeling of mrna expression levels during cns development and injury. In *PSB*, volume 4, pages 102–111, 1999.

[dJ02] Hidde de Jong. Modeling and simulation of genetic regulatory systems : a literature surveys. *Journal of Computational Biology*, 9(1) :67–103, 2002.

[Dje02] S. Djebali. Génération automatique de programmes tagcc par apprentissage. Master's thesis, LaMI UMR 8042,Univ. Evry, 2002.

[DS94] S. Dong and D. B. Searl. Gene structure prédiction by linguistic methods. *Genomics*, 23 :540–551, 1994.

[Dub02] A. L. Dubas. Etude d'un mécanisme de résolution par ordonnancement des solutions. Master's thesis, LaMI UMR 8042,Univ. Evry, 2002.

[ea95] T. E. Anderson et. al. A case for now (networks of workstations. *IEEE Micro*, 1995.

[Epi] Epigénèse. Atelier de la simulation en génomique vers l'épigénèse. `hhttp ://www.lami.univ-evry.fr/epigenese/`.

[FC96] J.L. Pazat F. Coelho, C.Germain. State of art in compiling hpf. In *The Data Parallel Programming Model*, volume LNCS 1132, pages 104–130. Springer, 1996.

[Fea91] P. Feautrier. Dataflow analysis of array and scalar references. *International Journal of Parallel Programming*, 20(1), Februar 1991.

[Fea92a] P. Feautrier. *Algorithmique parallèle*, chapter Techniques de parallélisation, pages 243–257. Masson, 1992.

[Fea92b] P. Feautrier. Some efficient solutions to the affine scheduling problem, ii, multidimensional time. *International Journal of Parallel Programming*, 21(6) :389–348, December 1992.

[Fic82] J. W. Fickett. Recognition of protein coding regions in dna sequences. *Nucleic acids research*, 10 :5303–5318, 1982.

[Fic96] J. W. Fickett. The gene identification problem : an overview for developpers. *Computers Chemistry*, 20(1) :103–118, 1996.

[For95] Message Passing Interface Forum. *MPI : A Message-Passing Interface Standard*, June 1995.

[Fos95] I. Foster. *Designign and Building Parallel Programs.* Addison-Wesley, 1995.

[FPA95] M. Le Fur, J-L. Pazat, and F. André. An array partitionning for parallel loop distribution. In *Europar*. Sringer Verlag, 1995.

[FR88] F.Irigoin and R.Triolet. Supernode partitionning. In *15 th ACM Symposium on Principles of Programming Languages*, pages 319–329, 1988.

[FTR02] Manolo Gouyb Fariza Tahi and Mireille Régnier. Automatic rna secondary structure prediction with a comparative approach. *Computers & Chemistry*, 26(5) :521–530, July 2002.

[FY97] C. Fu and T. Yang. Run-time techniques for exploiting irregular task parallelism on distributed memory architecture. *Journal of Parallel and Distributed Computing*, 42 :143–156, 1997.

[GBBK02] N. Guelzim, S. Bottani, P. Bourgine, and F. Képès. Topological and causal structure of the yeast transcriptional regulatory network. *Nature Genetics*, 31 :60–63, 2002.

[GBC03] J. Guespin, G. Bernot, and J-P. Comet. Les réseaux de régulation biologique : rencontre entre biologie etvinformatique. *TSI*, To appear - 2003.

[GD95] C. Germain and F. Delaplace. Automatic vectorization of communications for data-parallel programs. In *Euro-par 95*, number 966 in LNCS. Springer-Verlag, August 1995.

[GD97] C. Germain and F. Delaplace. compiling the block cyclic distributions. In *PARCO*, Bonn, Germany, September 1997.

[GDC94] C. Germain, F. Delaplace, and R. Carlier. A static execution model for data parallelism. *Parallel Processing Letter*, 4 :367–378, December 1994.

[Gia99] Jean-Louis Giavitto. *Scientific Repport for the HDR.* PhD thesis, LRI, Université de Paris-Sud, centre d'Orsay, September 1999. Research Report 1226.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and intractability.* Mathematical Sciences. Freeman, 1979.

[GKK97] A. Gupta, G. Karypis, and V. Kumar. Highly scalable parallel algorithms for sparse matrix factorization. *IEEE Transactions on Parallel and Distributed Systems*, 8(5) :502–520, May 1997.

[GM02] J.-L. Giavitto and O. Michel. The topological structures of membrane computing. *Fundamenta Informaticae*, 49 :107–129, 2002.

[GMD01] J. L. Giavitto, O. Michel, and F. Delaplace. Declarative simulation of dynamical systems : the 8,5 programming language and its application to the simulation of genetic networks. In *IPCAT*, 2001.

[GS90] M. T. Goodrich and J. S. Snoeyink. Stabbing parallel segments with a convex polygon. *Computer Vision, Graphics, and Image Processing*, 49(2) :152–170, February 1990.

[GUD95] M. Gengler, S. Ubéda, and F. Desprez. *Initiation au Parallélisme : Concepts, Architectures et Algorithmes.* Manuels informatiques. Masson, 1995. ISBN 2-225-85014-3.

[Gui99] R. Guigo. *DNA composition, codon usage and exon prediction*, pages 53–80. Academic Press, 1999.

[Gut84] A. Guttman. *R*-trees : a dynamic index structure for spatial searching. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 14(2) :47–57, 1984.

[HDS+95] Y.S Hwang, R. Das, J. Saltz, B. Brooks, and M. Hodoscek. Parallelizing molecular dynamics programs for distributed memory machines : An application of the chaos runtime support library. *IEEE Computational Science and Engineering*, 2(2) :18–29, 1995.

[Hil85] D. Hillis. *The Connection Machine.* MIT Press, 1985.

[HKK+91] S. Hiranandani, K. Kennedy, C. Koelbel, U. Kremer, and C.-W. Tseng. An overview of the Fortran D programming system. In U. Banerjee, D. Gelernter, A. Nicolau, and D. Padua, editors, *Languages and Compilers for Parallel Computing, Fourth International Workshop*, Santa Clara, CA, 1991. Springer-Verlag.

[HNP91] M.T. Heath, E. Ng, and B.W. Peyton. Parallel algorithm for sparse linear systems. *Siam Review*, 33(3) :420–460, September 1991.

[HPF] HPF Forum,
`http ://www.crpc.rice.edu/HPFF/home.html`. *High Performance Fortran.*

[IBS+02] J. Ihmels, S. Bergmann, O. Sarig, Y. Ziv, and N. Barkai. Revealing modular organization in the yeast transcriptional network. *Nature Genetics*, 31 :370–376, August 2002.

[JGB97] E. Johnson, D. Gannon, and P. Beckman. HPC++ : Experiments with the parallel standard template library. In *Proceedings of the 11th International Conference on Supercomputing (ICS-97)*, pages 124–131, New York, July 7–11 1997. ACM Press.

[JLGD03] O. Michel J.-L. Giavitto and F. Delaplace. Declarative simulation of dynamicals systems : the 8,5 programming language and its application to the simulation of genetic networks. *Biosystems*, 68 :155–170, March 2003.

[Kah74] G. Kahn. The semantics of a simple language for parallel programming. In *Proceeding IFIP Congress*, pages 471–475, 1974.

[Kau93] S.A. Kaufman. *The Origins of Order : Self-Organization and Selection in Evolution.* Oxford University Press, 1993.

[Kei85] J. M. Keil. Decomposing a polygon into simpler components. *SIAM Journal on Computing*, 14(4) :799–817, 1985.

[KHD02] T. Chothia K.L. Howe and R. Durbin. generic framework for the integration of gene prediction data by dynamic programming. *Genome Research*, 12(9) :1418–1427, 2002.

[Kit02] H. Kitano. Computational systems biology. *Nature*, 420(6912) :206–10, 2002.

[KMP98] S. Khanna, S. Muthukrishnan, and M. Paterson. On approximating rectangle tiling and packing. In *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 384–393, San Francisco, California, 25–27 January 1998.

[Koe91] C. Koelbel. Compile-time generation of communication for scientific programs. In *Supercomputing'91*, pages 101–110, 1991.

[KPS97] V. Kotlyar, K. Pingali, and P. Stoghill. Compiling parallel code for sparse matrix applications. In *SuperComputing.* ACM/IEEE, November 1997.

[LB98] Alexander V. Lukashin and Mark Borodovsky. Genemark. hmm : new solutions for gene finding. *Nucleic Acids Research*, 26(4) :1107–1115, februar 1998.

[Lew00] Benjamin Lewin. *Genes VII.* Oxford University Press, 2000.

[LFZRM98] G. Hartzell L. Florea, Z. Zhang, G. M. Rubin, and W. Miller. A computer program for aligning a cdna sequence with a genomic dna sequence. *Genome Research*, 8 :967–974, 1998.

[Li97] Wentian Li. The study of correlation structures of dna sequences : a critical review. *Computers Chem.*, 21(4) :257–271, 1997.

[LLL+79] W. Lipski, E. Lodi, F. Luccio, C. Mugnai, and L. Pagli. On two dimensional data organization II. *Fundamenta Informaticae*, 2 :245–260, 1979.

[Lop96] M. Lopez. Efficient decomposition of polygons into l shapes with applications to VLSI layouts. *ACM Transactions on Design Automation of Electronic Systems*, 1(3) :371–395, July 1996.

[LRRa02] T. H. Lee, N. J. Rinaldi, F. Robert, and al. Transcritional regulatory networks in *Saccharomyces cervisiae*. *Science*, 298 :799–804, October 2002.

[MDNM02] H. Matsuno, A. Doi, M. Nagasaki, and S. Miyano. Hybrid petri net representation of gene regulatory network. In *Pacific Symposium of Biocomputing*, pages 341–352, 2002.

[MGRH00] H. Tammana M. G. Reese, D. Kulp and D. Haussler. Genie-gene finding in drosophila melanogaster. *Genome Research*, 19 :529–538, 2000.

[Mic96] 0. Michel. Design and implementation of $81/2$, a declarative data-parallel language. *Computer Languages*, 22(2/3) :165–179, 1996. special issue on Parallel Logic Programming.

[MIP] MIPS. Mips database. `http ://www.mips.biochem.mpg.de/`.

[MJ94] Borodovsky M and McIninch J. Genmark : Parallel gene recognition for both dna strands. *Computer and Chemistry*, 17(2) :123–130, 1994.

[MM98] G. Marnellos and E. Mjolsness. A gene network approach to modeling early neurogenesis. In *PSB*, pages 30–41, 1998.

[Moh96] M. Mohri. On some applications of finite-state automata theory to natural language processing. *Journal of Natural Language Engineering*, 2 :1–20, 1996.

[Moh97] M. Mohri. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23(4), 1997.

[MPR00] M. Mohri, F. C. N. Pereira, and M. Riley. The design principles of a weighted finite-state transducer library. *Theoretical Computer Science*, 231(1) :17–32, 2000.

[MR90] M. Metcalf and J. Reid. *Fortran 90 Explained.* Oxford Uniersity Press, 1990.

[MS96] M. Mohri and R. Sproat. An efficient compiler for weighted rewrite rules. In *Meeting of the Association for Computational Linguistics*, pages 231–238, 1996.

[MSOI+02] R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon. Network motifs : Simple building blocks of complex networks. *Science*, 298 :824–827, 2002.

[MT98] K. Murakami and T. Takagi. Gene recognition by combination of several gene-finding programs. *Bioinformatics*, 14(9) :665–675, 1998.

[MU97] P.H. Zima M Ujaldon, E.L. Zappata B.M. Chapman. Vienna-fortran/hpf extension for sparse and irregular problems and their compilation. *IEE Trans. on Parallel & Distributed systems*, 8(10) :1068–1082, October 1997.

[Obj97] ObjectSpace, `http ://www.objectspace.com/jgl`. *JGL - The Generic Collection Library for Java*, 1997.

[Ope] OpenMp. Openmp : Simple, portable, scalable smp programming. `http ://www.openmp.org/`.

[OS83] J. O'Rourke and K. J. Supowit. Some NP-hard polygon decomposition problems. *IEEE Trans. Inform. Theory*, IT-29(2) :181–190, 1983.

[Paz97] J.L. Pazat. *Génération Automatique de code réparti, Habilitation à diriger des recherches*. PhD thesis, IRISA, 1997.

[PBG00] G. Parra, E. Blanco, and R. Guigo. Geneid in drosophila. *Genome Research*, 10 :511–515, 2000.

[PC] A. de Ricqlès P. Courrière, P. Delattre. *Structure et fonctions*, chapter XV. Encyclopedia Universalis, -.

[PC03] S. Pérès and J.-P. Comet. Contribution of computation tree logic to biological regulatory networks : example from pseudomonas aeruginosa. In C. Priami, editor, *Proceedings of the 1st Intern. Workshop CMSB'2003*, LNCS 2602, pages 47–56. Springer-Verlag, 2003.

[PHS+95] R. Ponnusamy, Y. Hwang, J. H. Saltz, A. Choudhary, and G. Fox. Supporting irregular distributions using data-parallel languages. *IEEE Parallel and Distributed Technology*, 3(1) :12–24, Spring 1995.

[Pug92] W. Pugh. A practical algorithm for exact dependence analysis. *Communications of the ACM*, 35(8) :102–114, August 1992.

[Ram92] J. Ramajunan. Non-unimodular transformations of nested loops. *IEEE Computer*, 16(20), 1992.

[RD00a] D. Remy and F. Delaplace. Paradeis : an extension of stl for sparse parallel computations. In Vecpar, editor, *Vecpar 2000*, 2000.

[RD00b] D. Remy and F. Delaplace. Paradeis : une extension de stl pour le calcul parallèle creux. *TSI*, 19(9) :269–1298, November 2000.

[Rem00] D. Remy. *Paradeis : un environnement parallèle pour le traiteme du creux*. PhD thesis, U.E.V.E., 2000.

[RG00] P. Agarwal & al R. Guigo. An assessment of genes prediction accuracy in large dna sequence. *Genome Research*, 10 :1631–1642, 2000.

[Riv01] L. Rival. Indentification de critères efficaces pour l'annotation *ab-initio* en tagcc. Master's thesis, LaMI UMR 8042,Univ. Evry, 2001.

[RS97] G. Rozenberg and A. Salomaa. *Handbook of Formal Languages*, volume 1. Springer Verlag, Berlin, Heidelberg, New York., 1997.

[RS00] X. Wen R. Somogyi, S. Fuhrman. *Genetic Network Inference in Computationnal Models and Application to Large Scale Expression*, chapter 5, pages 199–157. MIT Press, 2000.

[Sea93] D. B. Searls. String variable grammar : A logic grammar formalism for the biological language of dna. *Journal of Logic Programming*, 24(1&2) :73–102, 1993.

[Sea94] B. Searl. Gene structure prediction by linguistic methods. *Genomics*, 23 :540–551, 1994.

[Sea02] D. B. Searls. The language of genes. *Nature*, 420(6912) :211–217, Nov. 2002.

[SH89] G. Stormo and G. Hartzell. Identifying protein-binding sites from unaligned dna fragments. In *Proceedings of the National Academy of Science USA*, volume 89, pages 1183–1187, February 1989.

[Shr86] A. Shrijver. *Theory of linear and Integer Programming.* J.Wisley & Sons, 1986.

[SL94] A. A. Stepanov and M. Lee. The Standard Template Library. Technical Report X3J16/94-0095, WG21/N0482, ISO Programming Language C++ Project, May 1994.

[SMC91] J.H. Saltz, R. Mirchandaney, and K. Crowley. Run-time parallelization and scheduling of loops. *IEEE Transaction on Computer*, 40(5) :603–611, May 1991.

[SS00] A. Salamov and V. Solovyev. Ab initio gene finding in drosophila genomic dna. *Genome Research*, 10 :516–522, 2000.

[SSOA02] S. Mangan S. Shen-Orr, R. Milo and U Alon. Network motifs in the transcriptional regulation network of escherichia coli. *Nature Genetics*, 31 :64–68, 2002.

[SSR+03] E. Segal, M. Shapira, A. Regev, D. Pe'er, Botstein D., and Koller D. Module networks : indentiflying regulatory modules and their condition-specific regulators from gene expression data. *Nature Genetics*, 34(2) :166–176, 2003.

[Sta84] R. Staden. Computer methods to locate signals in nucleic acid sequences. *Nucleic Acids Research*, 12 :505–519, 1984.

[Ste97] Robert Stephens. A survey of stream processing. *Acta Informatica*, 34(7) :491–541, 1997.

[Sto97] P. Stodghill. *A Relational Approach to the Automatic Generation of Sequential Sparse Matrix Codes.* PhD thesis, Cornell University, 1997.

[Str99] B. Stroustrup. *Le langage C++.* CampusPress, troisième edition, 1999.

[Str01] G. Strong. Information, the language of biology. Language Modeling Data Workshop, mars 2001.

[Tag] Tagcc. Tagcc. `http ://tagcc.lami.univ-evry.fr/`.

[Tho91] Thomas. Regulatory networks seen as asynchronous automata : A logical description. *J. theor. Biol.*, 153, 1991.

[TR99] D. Thieffry and D. Romero. The modularity of biological regulatory networks. *Biosystems*, 50 :49–59, April 1999.

[Tse93] C-W. Tseng. *An optimizing Fortran D compiler for MIMD distributed-memory machines.* PhD thesis, Rice University, 1993.

[TSRT95] D. Thieffry, E.L. Snoussi, J. Richelle, and R. Thomas. Positive loops and differentiation. *J. Biol. Systems*, 3(2) :457–466, 1995.

[TT95] R. Thomas and D. Thieffry. Dynamical behaviours of regulatory networks - II . immunity control in bactériophage lambda. *Bulletin of Mathematical Biology*, 57(2) :277–297, 1995.

[TTK95] R. Thomas, D. Thieffry, and M. Kaufman. Dynamical behaviours of regulatory networks - I . biological role of feedback loops and pratical use of the concept of feedback loop. *Bulletin of Mathematical Biology*, 57(2) :247–276, 1995.

[UZ95] M. Ujaldon and E. L. Zapata. Efficient resolution of sparse indirections in data-parallel compilers. In *Proc. International Conference on Supercomputing*, pages 117–126, Barcelona, July 1995.

[Val90] L. G. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8) :103, August 1990.

[vHKK+93] R. von Hanxleden, K. Kennedy, C. Koelbel, R. Das, and J. Saltz. Compiler analysis for irregular problems in Fortran D. *LNCS*, 757 :97–111, 1993.

[Voh01a] J. Vohradsky. Neural model of gene expression. *Faseb*, 15 :846–854, 2001.

[Voh01b] J. Vohradsky. Neural model of genetic network. *Journal of Biology and Chemistry*, 276 :38168–36173, 2001.

[Wat89] M. S. Waterman. *Mathematical Methods for DNA Sequences*, chapter Sequence Alignments, pages 53–92. Boca Raton, 1989.

[WDS+95] J. Wu, R. Das, J. Saltz, H. Berryman, and S. Hiranandani. Distributed memory compiler design for sparse problems. *IEEE Transactions on Computers*, 25(6) :737–753, June 1995.

[Wol96] M. Wolfe. *High Performance Compilers for Parallel Computing*. Addison Wesley, 1996.

[WWS99] D. C. Weaver, C. T. Workman, and G. D. Stormo. Modeling regulatory networks with weight matrices. In *PSB*, volume 4, pages 112–123, 1999. neural network modeling.

[ZO02] A-L. Barabasi ZN. Oltvai. Life's complexity pyramid. *Science*, 298 :763–764, 2002.