Numéro d'ordre : 2907

Université des Sciences et Technologies de LILLE

THÈSE

Spécialité

INFORMATIQUE

Présentée par

Emmanuel CAGNIOT

En vue de l'obtention du grade de
Docteur de l'Université des Sciences et Technologies de LILLE

Algorithmes Data-parallèles Irréguliers Appliqués à la Simulation Électromagnétique Tridimensionnelle

Soutenue le 20 décembre 2000
devant le Jury composé de :

M. Serge PETITON	Président	LIFL
M. Jean ROMAN	Rapporteur	LaBRI
M. Laurent NICOLAS	Rapporteur	CEGELY
M. Thomas BRANDES	Examinateur	GMD-SCAI
Mlle Marie Christine COUNHIL	Examinatrice	LaBRI
M. Jean-Luc DEKEYSER	Examinateur	LIFL
M. Francis PIRIOU	Examinateur	L2EP
M. Adel RAZEK	Examinateur	LGEP

Remerciements

Avant toute chose, je tiens à remercier deux personnes sans lesquelles ce travail n'aurait pu avoir lieu :

- Gil UTARD, Maître de Conférences au Laboratoire de Recherche en Informatique d'Amiens, qui, lorsqu'il était mon co-responsable de stage de DESS à l'École Normale Supérieure de LYON, m'a convaincu de commencer un travail de thèse. Je tiens à le remercier non seulement pour m'avoir fait confiance, mais également pour m'avoir recommandé auprès de Jean-Luc DEKEYSER. J'espère qu'il ne sera pas déçu en lisant ces pages. En tout cas, j'ai fait de mon mieux ;
- Thomas BRANDES, qui a également accepté d'être examinateur dans mon jury et sans qui rien de ce travail n'aurait été possible. Je tiens à le remercier pour toute l'aide apportée, que ce soit au niveau du développement grâce aux différentes versions de son compilateur HPF ADAPTOR, ou au niveau des publications. J'ai énormément apprécié de travailler avec lui dans ces conditions, moi en tant qu'utilisateur, lui en tant que développeur de compilateurs HPF, ce qui a permis de croiser les points de vue et de valider certains concepts. Je tiens à m'excuser pour tous les problèmes que je lui ai causés avec mon application.

Je remercie Jean-Luc DEKEYSER et Francis PIRIOU, mes deux directeurs de thèse, non seulement pour m'avoir confié ce travail mais également pour m'avoir fait confiance en me laissant une très grande marge de liberté. Je tiens à m'excuser auprès des deux pour ma mauvaise humeur de certains jours ("c'est comme ça, c'est comme à l'armée ! ...") et auprès de Francis PIRIOU pour ne pas lui avoir avoué tout de suite que durant mon DEUG, l'électromagnétisme était pour moi un mystère et qu'il m'arrivait régulièrement de calculer des inductions magnétiques de l'ordre de quelques milliers de Tesla ...

Je remercie Pierre BOULET et Stéphane CLÉNET, qui, bien que co-encadrants non-officiels, ne m'en ont pas moins beaucoup aidé :

- Pierre pour HPF, les publications et la re-lecture (les re-lectures) de mon mémoire ;
- Stéphane pour tous mes problèmes en électromagnétisme et plus spécialement sur la discrétisation des termes sources.

Je tiens également à remercier :

- Serge PETITON, non seulement pour avoir accepté de présider mon jury mais également pour m'avoir encadré lors d'un projet de DESS qui avait pour objectif la parallélisation de la méthode d'Arnoldi. Ce projet m'a

permis de comprendre le principe des méthodes de Krylov et fait gagner un temps précieux lorsque j'ai abordé les préconditionneurs par blocs et les méthodes de sous-domaines ;
- Jean ROMAN et Laurent NICOLAS pour avoir accepté de rapporter mon mémoire de thèse et pour avoir envoyé leur rapport dans les temps alors que je n'y étais pas ;
- Marie Christine COUNHIL et Adel RAZEK pour leur présence à mon jury en tant qu'examinateurs. Je tiens à remercier particulièrement Marie Christine COUNHIL pour ses nombreuses remarques et conseils concernant HPF 2.0 et d'autres approches possibles pour les applications irrégulières.

Enfin, je tiens à remercier :
- toutes les personnes avec qui j'ai partagé le bureau 326 et sa cafetière tout au long de ces trois années. J'espère que Florent DEVIN, qui ne me lache pas depuis le DESS, comprendra enfin qu'à XBLAST c'est moi le plus fort et que ce n'est pas en posant des dynamites partout que l'on gagne une partie. Un grand merci à Julien SOULA pour ses sauvetages lors de mes nombreux naufrages en Shell et en macro LATEX ;
- tous les habitués de la salle de calcul du L2EP, notamment Yvonnick LE MENACH et Georges MARQUES pour l'aide apportée dans mon travail ;
- les personnels de la section SICF du CFAT où je suis affecté en tant qu'officier de réserve (je n'ai pas complètement réussi à raccrocher le treillis) pour les bons moments passés à SISSONNE et MOURMELON et les formations réseaux, domaine où il y a quelques temps encore, je ne connaissais rien de pratique.

Je vaux ce que je veux.

Table des matières

Introduction générale

Comme dans d'autres secteurs de l'industrie, la conception et la réalisation d'un dispositif électromagnétique sont basées sur un processus itératif de prototypage. La phase de conception consiste à définir un comportement et des plans de fabrication. Le prototypage permet de confirmer ou d'infirmer certaines hypothèses de la phase de conception, en construisant et en testant une pré-version du dispositif. Les résultats obtenus sont ensuite ré-injectés dans la phase de conception afin d'établir de nouveaux plans de fabrication, c'est pourquoi nous parlons de processus itératif. Le dispositif est considéré comme réalisé lorsque le comportement du prototype est suffisamment proche de celui voulu par la phase de conception. De par son unicité, l'usinage d'un prototype tel que celui d'une machine électrique, est une opération longue et coûteuse. De plus, aucun nouveau plan de fabrication ne peut être établi avant la fin de la période d'essai. Dès lors, le processus itératif employé est économiquement onéreux.

La simulation numérique est une solution permettant de répondre à ce problème. Son but est de prédire le comportement d'un dispostif sur ordinateur à partir d'un modèle mathématique donné. La qualité de la prédiction dépend de la finesse du modèle employé. Cette prédiction permet de faire l'économie de bon nombre de prototypes, en confirmant ou en infirmant certaines hypothèses de la phase de conception avant la phase d'usinage. L'utilisation de la simulation numérique permet ainsi de réduire les coûts de développement du dispositif considéré.

Il est possible de modéliser de nombreux dispositifs électromagnétiques à partir d'un modèle mathématique bi-dimensionnel, en privilégiant une direction pour le courant. Cependant, certains dispositifs, du fait de la complexité de leur géométrie ou des effets de leurs extrêmités, nécessitent un modèle mathématique tridimensionnel. Le problème qui se pose alors est celui de l'explosion du nombre d'inconnues, problème encore aggravé par les phénomènes de non-linéarité et d'évolutivité dans le temps. Dans ces conditions, les limites des stations de travail en termes de puissance processeur et de capacité de stockage mémoire, sont rapidement atteintes. Dès lors, afin d'effectuer des simulations proches de la réalité en des temps acceptables, le recours aux machines parallèles doit être envisagé.

Parmi les calculateurs parallèles, la communauté scientifique non-spécialiste connaît et apprécie généralement les machines vectorielles. Pour les exploiter, le programmeur ajoute des directives dans l'application qu'il désire vectoriser. Ces directives sont des informations sur le code adressées au compilateur. Lorsque ces informations sont insuffisantes, le compilateur le signale au programmeur

en l'invitant à modifier la zone concernée ou à donner un complément d'information. Il s'agit donc d'un processus à la fois simple et rassurant pour le programmeur, ce dernier ne modifiant ni la structure de son application, ni ses algorithmes fondamentaux.

Bien que la parallélisation automatique ait beaucoup progressé [11, 12], la complexité atteinte par les applications de simulation numérique est telle qu'une simple vectorisation ne suffit plus. Il est alors nécessaire d'envisager une parallélisation manuelle de ces applications. Ce type de parallélisation réclame une bonne connaissance des mécanismes fondamentaux des machines parallèles, des différents outils, langages et bibliothèques utilisables, des méthodes numériques dédiées au parallélisme ainsi qu'une certaine dextérité en matière de programmation. Cette connaissance et cette dextérité ne contribuent généralement pas à rendre la programmation parallèle abordable pour les non-spécialistes, ces derniers se voyant également obligés de remettre en cause leurs connaissances en matière d'algorithmique, un bon algorithme séquentiel pouvant se révéler être un mauvais algorithme parallèle.

Le développement d'une application parallèle est souvent confié à un spécialiste. Son rôle consiste tout d'abord à s'immerger dans le domaine concerné par l'application. À partir des connaissances acquises sur ce domaine, de l'étude des algorithmes utilisés et des machines parallèles concernées, il lui est alors possible de définir les différents types de parallélisme à employer ainsi que les outils les mieux adaptés. Toute la difficulté de la programmation parallèle réside dans ce choix. Lorsque l'application concernée est peu évolutive ("boite noire"), le meilleur choix est celui de l'efficacité au détriment des critères de qualité du génie logiciel. À l'inverse, lorsque cette application est en constante mutation, un compromis entre les impératifs d'efficacité et les critères du génie logiciel doit être trouvé, afin que la maintenance puisse être assurée par des non-spécialistes. De tels compromis ont lieu dans d'autres domaines de l'informatique. Ainsi, dans le cas des systèmes d'exploitation, la grande majorité des fonctionnalités est écrite dans un langage de haut niveau, généralement le C. Les fonctionnalités critiques telles que celles assurant les accès disques ou le bootstrap du système, sont très minoritaires et bien souvent figées. Elles sont généralement regroupées dans une zone très restreinte du code du système d'exploitation concerné. Contrairement aux autres fonctionnalités, les fonctionnalités critiques sont écrites en langage assembleur, ce qui leur confère une efficacité maximale, efficacité qu'un langage de haut niveau ne peut atteindre.

Curieusement, dans le domaine de la simulation numérique, le choix de l'efficacité l'emporte bien souvent sur toute idée de compromis. À cela, nous pouvons imaginer plusieurs raisons. Tout d'abord, le langage de programmation employé est généralement le Fortran 77. Or, il est de notoriété publique qu'un tel langage ne dispose d'aucune caractéristique (types définis par le programmeur, modules, etc.) permettant au programmeur d'appliquer les concepts du génie logiciel. Toute idée de compromis suppose un mélange de plusieurs outils au sein d'une même application. Ces outils n'étant pas toujours conçus pour une telle utilisation, ce mélange engendre parfois des incompatibilités. La tentation du "tout-homogène" permet d'éviter ce problème en n'utilisant qu'un seul de ces outils. Dans cette hypothèse, les outils de haut niveau ne permettent générale-

ment pas d'écrire une application dans son intégralité (l'exemple du langage C dans les systèmes d'exploitation). Il peut alors être tentant d'utiliser des outils de plus bas niveau, mais qui permettent d'écrire l'intégralité du code, qui plus est de manière efficace.

Le travail que nous présentons dans ce mémoire est une approche parallèle de haut niveau pour ce type d'application, approche basée sur l'idée de compromis entre les impératifs d'efficacité et les critères de qualité du génie logiciel. L'application concernée est un code de recherche basé sur les éléments finis, écrit en Fortran 77 et permettant de modéliser des dispositifs électromagnétiques en trois dimensions. L'aspect génie logiciel est pris en charge par la nouvelle norme qu'est Fortran 90 [45] et par le modèle de programmation à parallélisme de données incarné par le langage standard High Performance Fortran [38]. Comme pour l'exemple des systèmes d'exploitation évoqué précédemment, HPF ne permet pas d'écrire l'intégralité de cette application, en raison de la forte irrégularité de cette dernière. Cette irrégularité est le corollaire des méthodes de discrétisation telles que la méthode des éléments finis et résulte de l'utilisation de matrices creuses dont les caractéristiques ne peuvent être établies qu'au moment de l'exécution. Dans ces conditions, la carte des communications inter-processeurs de l'application ne peut être déterminée par HPF au moment de la compilation, ce qui se traduit par une gestion inefficace de ces dernières. Pour rendre ces communications efficaces, nous faisons appel aux bibliothèques de passage de messages. Ces outils sont le propre d'un autre modèle de programmation parallèle : le modèle à parallélisme de tâches. Il s'agit d'un concept de plus bas niveau que le modèle à parallélisme de données, mais réputé pour son efficacité. Les bibliothèques de passage de messages que nous utilisons ne sont pas les bibliothèques standard PVM [35] ou MPI [36] car nous les considérons de trop bas niveau pour être abordables par des non-spécialistes, mais des bibliothèques conçues pour être intégrées dans les applications data-parallèles irrégulières [3].

La première partie de ce mémoire présente une étude bibliographique sur le sujet. Le premier chapitre de cette partie présente les principes de base ainsi que les différentes formulations des problèmes utilisés dans la modélisation électromagnétique. Les formulations indépendantes du temps sont appelées formulations magnétostatiques tandis que celles qui dépendent du temps sont appelées formulations magnétodynamiques. La résolution analytique de ces formulations n'est généralement pas possible et le recours aux méthodes de discrétisation doit être envisagé. Celle que nous utilisons est la méthode des éléments finis. Elle suppose tout d'abord la définition d'une structure mathématique du niveau continu permettant de représenter les champs et les potentiels (complexe de De Rham), puis la définition d'une structure mathématique analogue au niveau discret (complexe de Whitney). Les éléments finis de Whitney [9] sont des éléments finis mixtes conçus pour l'électromagnétisme et qui comprennent les éléments finis nodaux, d'arêtes, de facettes et de volumes. Ils permettent de discrétiser les différentes formulations utilisées et d'obtenir les systèmes d'équations correspondants. Le second chapitre de cette partie contient une présentation générale du parallélisme. Y sont abordés les différents types de machines parallèles, les différents modèles d'exécutions et de programmation ainsi que les différents critères d'évaluation d'une application parallèle. Nous abordons également, de manière

générale, l'évolution actuelle des machines parallèles autour de la notion de SMP (Symmetric Multi Processors).

Enfin, le dernier chapitre de cette partie présente les méthodes de Schwarz, dont les représentantes les plus connues sont les méthodes de sous-domaines. Les méthodes de Schwarz permettent de construire des matrices par blocs et des préconditionneurs associés. Ce type de structure est parfaitement adapté aux architectures parallèles et autorise une résolution itérative efficace des systèmes d'équations correspondants. Les méthodes de Schwarz s'appliquent naturellement aux systèmes d'équations résultant de la discrétisation des formulations magnétostatiques. La décomposition de Helmholtz permet de montrer qu'il en est de même pour les formulations magnétodynamiques.

La seconde partie de ce mémoire présente une approche de haut niveau pour la parallélisation de notre application. Le premier chapitre de cette partie présente une parallélisation très proche des méthodes de vectorisation, en ce sens que nous parallélisons un code existant sans rien changer de sa structure et de ses algorithmes fondamentaux, c'est pourquoi nous la qualifions d'approche parallèle légère. Après avoir présenté les caractéristiques et les algorithmes de l'application, nous présentons les contraintes liées à cette dernière, les différents types de parallélisme à mettre en œuvre ainsi que le modèle de programmation et les outils qui nous paraissent les mieux adaptés. Ce modèle et ces outils sont respectivement le modèle de programmation à parallélisme de données, Fortran 90, High Performance Fortran et Halos [3], une bibliothèque de passage de messages de haut niveau conçue pour les applications irrégulières basées sur le modèle à parallélisme de données. Les formulations magnétostatiques et magnétodynamiques ne différant principalement que par l'ajout d'une boucle temporelle, nous nous focalisons sur la parallélisation de la partie magnétostatique de l'application en écrivant une version Fortran 90 optimisée et en la parallélisant avec le couple HPF-Halos. Le second chapitre de cette partie présente une parallélisation beaucoup plus poussée et efficace que celle présentée dans le chapitre précédent. Cette nouvelle approche tire les enseignements de l'approche précédente en utilisant les méthodes de Schwarz. Ces dernières imposent cependant de modifier la structure générale et les algorithmes fondamentaux de l'application, c'est pourquoi nous parlons d'approche parallèle lourde, par opposition à l'approche parallèle légère du chapitre précédent. Afin de préserver la facilité de maintenance de l'application, nous poussons plus loin le concept de génie logiciel en écrivant une nouvelle version Fortran 90 dont les différentes parties sont classées en zones évolutives et peu évolutives. Les zones peu évolutives sont regroupées et isolées dans des modules dont l'accès s'effectue via des interfaces imposées. Les zones évolutives utilisent, quant à elles, des algorithmes aussi similaires que possibles à ceux de l'approche précédente. La nouvelle version Fortran 90 étant écrite en fonction de la manière dont nous voulons la paralléliser avec le couple HPF-Halos, cette dernière étape est beaucoup plus facile à réaliser que dans l'approche précédente.

Nous terminons ce mémoire en présentant nos conclusions sur le travail effectué et ce qui nous paraît être les suites à donner à ce dernier.

Première partie

État de l'art

Chapitre 1

Modélisation électromagnétique

1.1 Introduction

La modélisation des problèmes électromagnétiques s'appuie sur les équations de Maxwell. Ces équations aux dérivées partielles font intervenir des champs physiques (champs électriques, champs magnétiques et autres). L'étude de ces champs en régime statique (indépendant du temps) constitue les modèles électrostatiques, électrocinétiques et magnétostatiques. Leur étude en régime dynamique (dépendant du temps), dans le cadre des hypothèses de l'électrotechnique (courant de déplacement négligé), constitue le modèle magnétodynamique. La section 1.2 présente le système de Maxwell et différentes formulations, équivalentes entre elles, des problèmes magnétostatiques et magnétodynamiques.

La résolution analytique des équations de Maxwell n'est possible que dans des cas particuliers, généralement simplifiés. Leur résolution dans le cas général nécessite de recourir à des méthodes numériques basées sur des techniques de discrétisation. Ces méthodes transforment les équations aux dérivées partielles en système d'équations algébriques. La résolution de ce système fournit une approximation de la solution recherchée [47].

Les formulations de la section 1.2 sont exprimées sous forme forte. Elles requièrent une certaine régularité de la solution qui les rend difficiles à discrétiser. La forme faible (ou forme intégrale) est une formulation plus "laxiste" que la forme forte, mais qui se prête mieux à la discrétisation [67]. Pour l'obtenir, nous devons construire des espaces fonctionnels pour les champs et les potentiels associés. Avec les opérateurs différentiels gradient, rotationnel et divergence, ces espaces fonctionnels forment une structure mathématique continue qui permet d'exprimer rigoureusement les équations de Maxwell. La section 1.3 présente cette structure encore appelée complexe de De Rham [9, 48].

Les méthodes de différences finies, éléments finis et volumes finis sont des méthodes de discrétisation. Leur principe est de transformer un problème continu en un problème discret analogue mais caractérisé par un nombre fini d'inconnues appelées degrés de liberté. Cette transformation passe par le remplacement des espaces fonctionnels continus impliqués par des espaces discrets. Ces derniers sont appelés espaces d'approximation.

Les éléments finis de Whitney [8, 9] permettent de conserver certaines pro-

priétés du niveau continu au niveau discret. Les espaces d'éléments finis qu'ils engendrent, forment, avec les équivalents discrets des opérateurs différentiels gradient, rotationnel et divergence, une structure mathématique discrète analogue à la structure mathématique continue. Elle permet d'exprimer les équations de Maxwel de manière tout aussi rigoureuse. La section 1.4 présente la méthode des éléments finis et cette structure discrète encore appelée complexe de Whitney.

En électromagnétisme, des jauges sont souvent nécessaires pour garantir l'unicité des solutions (différentes jauges sont présentées dans la section 1.2). Récemment, Ren [54] a présenté un cadre dans lequel ces jauges ne sont pas indispensables. La section 1.5 présente ce cadre et les formes discrétisées des formulations de la section 1.2. La discrétisation des formulations magnétodynamiques fait intervenir des couplages entre formulations magnétodynamiques et magnétostatiques.

Par la suite, les notations suivantes seront employées :

$\mathbf{a}$	vecteur (en gras)
b	scalaire
rot	opérateur vectoriel (en gras)
$grad$	opérateur scalaire
$\otimes$	produit vectoriel
$\odot$	produit scalaire
$\partial_x f$	dérivée partielle de f par rapport à x
A^T	transposée de la matrice A

1.2 Formulations des problèmes

Dans cette section, nous commençons par présenter le problème initial constitué des équations de Maxwell, des lois de comportement et des conditions de transmission associées. Puis, partant de ce problème initial, nous introduisons une variable mathématique, scalaire ou vectorielle, appelée potentiel (scalaire ou vecteur) et destinée à rendre implicite l'une des équations de Maxwell. Cette variable est généralement reliée à un champ physique par un opérateur vectoriel. L'introduction de ce potentiel nous permet de présenter différentes formulations, équivalentes entre elles, des problèmes magnétostatiques et magnétodynamiques.

Avant de présenter le système de Maxwell et les différentes formulations magnétostatiques et magnétodynamiques, nous rappelons qu'un domaine D est dit simplement connexe si tout contour simple qui y est tracé peut être le support d'une surface entièrement contenue dans D. Dans le cas contraire, le domaine est dit multiplement connexe. L'utilisation d'un potentiel scalaire n'est autorisée que dans un domaine simplement connexe. Par ailleurs, les rotationnels de deux fonctions vectorielles étant égaux à une fonction gradient près, une jauge est nécessaire pour garantir l'unicité de la solution.

1.2.1 Équations de Maxwell

Les équations de Maxwell forment un système d'équations aux dérivées partielles régissant l'ensemble des phénomènes électromagnétiques :

$$\mathbf{rot}\,\mathbf{h} = \mathbf{j} + \partial_t \mathbf{d} \tag{1.2.1}$$
$$\mathbf{rot}\,\mathbf{e} = -\partial_t \mathbf{b} \tag{1.2.2}$$
$$\mathbf{div}\,\mathbf{b} = 0 \tag{1.2.3}$$
$$\mathbf{div}\,\mathbf{d} = \rho \tag{1.2.4}$$

avec :

$\mathbf{h}$	le champ magnétique $(A \cdot m^{-1})$
$\mathbf{b}$	l'induction magnétique (T)
$\mathbf{e}$	le champ électrique $(V \cdot m^{-1})$
$\mathbf{d}$	l'induction électrique $(C \cdot m^{-2})$
$\mathbf{j}$	la densité de courant de conduction $(A \cdot m^{-2})$
ρ	la densité volumique de charges $(C \cdot m^{-3})$

À partir des équations 1.2.1 et 1.2.4, nous obtenons l'équation de conservation de la charge électrique :

$$\mathbf{div}\,\mathbf{j} + \partial_t \rho = 0 \tag{1.2.5}$$

Le terme $\partial_t \mathbf{d}$ de l'équation 1.2.1 représente le courant de déplacement. Les hypothèses de l'électrotechnique le considèrent négligeable devant le courant de conduction. Dès lors, nous avons :

$$\mathbf{rot}\,\mathbf{h} = \mathbf{j} \tag{1.2.6}$$
$$\mathbf{div}\,\mathbf{j} = 0 \tag{1.2.7}$$

Lois de comportement

Les inductions et les champs sont reliés entre eux par les lois de comportement qui expriment les propriétés des matériaux :

$$\mathbf{b} = \mu(\mathbf{h})\,\mathbf{h} \tag{1.2.8}$$
$$\mathbf{d} = \epsilon\,\mathbf{e} \tag{1.2.9}$$
$$\mathbf{j} = \sigma\,\mathbf{e} \tag{1.2.10}$$

avec :

μ	la perméabilité magnétique $(H \cdot m^{-1})$
ϵ	la permittivité électrique $(F \cdot m^{-1})$
σ	la conductivité électrique $(\Omega^{-1} \cdot m^{-1})$

Un matériau est dit non-linéaire lorsque sa perméabilité magnétique μ est dépendante du champ magnétique $\mathbf{h}$.

Conditions de transmission

Les champs subissent des discontinuités à l'interface entre deux milieux de propriétés différentes. Les conditions de transmission permettent d'exprimer des relations de continuité sur les composantes de ces champs :

$$\mathbf{n} \otimes (\mathbf{h_2} - \mathbf{h_1})_\Gamma = \mathbf{j_s} \tag{1.2.11}$$

$$\mathbf{n} \otimes (\mathbf{e_2} - \mathbf{e_1})_\Gamma = 0 \tag{1.2.12}$$

$$\mathbf{n} \odot (\mathbf{b_2} - \mathbf{b_1})_\Gamma = 0 \tag{1.2.13}$$

$$\mathbf{n} \odot (\mathbf{d_2} - \mathbf{d_1})_\Gamma = \rho_s \tag{1.2.14}$$

avec :

$\mathbf{j_s}$	la densité surfacique de courant
ρ_s	la densité surfacique de charges
Γ	l'interface entre les deux milieux
$\mathbf{n}$	la normale à l'interface Γ

Les grandeurs $\mathbf{j_s}$ et ρ_s n'interviennent pas dans les exemples traités. Dès lors, ces équations expriment respectivement la continuité de la composante tangentielle des champs magnétiques et électriques, et la continuité de la composante normale des inductions magnétiques et électriques.

La continuité de la composante normale de la densité de courant est obtenue à partir de l'équation 1.2.7 :

$$\mathbf{n} \odot (\mathbf{j_2} - \mathbf{j_1})_\Gamma = 0 \tag{1.2.15}$$

1.2.2 Formulations magnétostatiques

La magnétostatique est l'étude des phénomènes magnétiques en régime stationnaire. En notant $\mathbf{j_0}$ la densité de courant, les équations à considérer sont :

$$\mathbf{rot\ h} = \mathbf{j_0} \tag{1.2.16}$$

$$\mathbf{div\ b} = 0 \tag{1.2.17}$$

auxquelles il faut ajouter la loi de comportement :

$$\mathbf{b} = \mu\, \mathbf{h} \tag{1.2.18}$$

Formulation en potentiel scalaire magnétique

Pour prendre en compte les sources de courant présentes dans le domaine d'étude, nous pouvons décomposer le champ magnétique en deux parties $\mathbf{h} = \mathbf{h_s} - grad\ \Omega$. Le champ source $\mathbf{h_s}$ permet de vérifier l'équation 1.2.16 tandis que le potentiel scalaire magnétique Ω permet d'introduire les champs à rotationnel nul. À partir des équations 1.2.17 et 1.2.18, nous obtenons alors :

$$\mathbf{div}\ (\mu\ (\ \mathbf{h_s} - grad\ \Omega)\ ) = 0 \tag{1.2.19}$$

La continuité de la composante normale de l'induction magnétique est implicite dans l'équation 1.2.19. La continuité de la composante tangentielle du champ magnétique est assurée par le choix de $\mathbf{h_s} - grad\ \Omega$.

Formulation en potentiel vecteur magnétique

L'équation 1.2.17 fait que l'induction magnétique dérive d'un potentiel vecteur magnétique **a** tel que $\mathbf{b} = \mathbf{rot}\,\mathbf{a}$. À partir de l'équation 1.2.16, nous obtenons alors :

$$\mathbf{rot}\,(\nu\,\mathbf{rot}\,\mathbf{a}) = \mathbf{j}_0 \tag{1.2.20}$$

avec :

ν	l'inverse de la perméabilité magnétique

La continuité de la composante tangentielle du champ magnétique est implicite dans l'équation 1.2.20. La continuité de la composante tangentielle du potentiel vecteur magnétique assure la continuité de la composante normale de l'induction magnétique.

Parmi les jauges possibles pour le potentiel vecteur **a**, citons la jauge de Coulomb $\mathbf{div}\,\mathbf{a} = 0$ et la jauge $\mathbf{a} \odot \mathbf{w} = 0$ dans laquelle **w** est un champ vectoriel dont les lignes ne se referment pas [2].

1.2.3 Formulations magnétodynamiques

La magnétodynamique est l'étude des phénomènes magnétiques instationnaires dans le cadre des hypothèses de l'électrotechnique. Les équations à considérer sont :

$$\mathbf{rot}\,\mathbf{h} = \mathbf{j} \tag{1.2.21}$$
$$\mathbf{rot}\,\mathbf{e} = -\,\partial_t \mathbf{b} \tag{1.2.22}$$
$$\mathbf{div}\,\mathbf{b} = 0 \tag{1.2.23}$$
$$\tag{1.2.24}$$

auxquelles il faut ajouter les lois de comportement :

$$\mathbf{b} = \mu\,\mathbf{h} \tag{1.2.25}$$
$$\mathbf{j} = \sigma\,\mathbf{e} \tag{1.2.26}$$

Le traitement des problèmes électromagnétiques autorise la manipulation directe des champs physiques. Cependant, le problème de la discontinuité des composantes normales à l'interface entre deux milieux de propriétés différentes peut être solutionné par l'introduction de potentiels. En effet, l'équation 1.2.23 fait que l'induction magnétique dérive d'un potentiel vecteur magnétique **a** tel que $\mathbf{b} = \mathbf{rot}\,\mathbf{a}$. L'équation 1.2.22 permet alors d'écrire $\mathbf{e} = -\,(\partial_t \mathbf{a} + \mathit{grad}\,\varphi)$ où φ est un potentiel scalaire électrique. En définissant un potentiel vecteur électrique **t** tel que $\mathbf{rot}\,\mathbf{t} = \mathbf{j}$, l'équation 1.2.21 permet d'écrire $\mathbf{h} = \mathbf{t} - \mathit{grad}\,\Omega$ où Ω est un potentiel scalaire magnétique. Dès lors, nous constatons que de par la définition des potentiels **a** et **t**, la continuité des composantes normales de **b** et **j** est assurée.

Les formulations magnétodynamiques présentées ne sont valables que dans les régions conductrices. Les termes ν et η désignent respectivement les inverses de la perméabilité magnétique et de la conductivité électrique. Les jauges employées sont les mêmes que pour les formulations magnétostatiques.

Formulation en potentiels a-φ

Cette formulation utilise le potentiel vecteur magnétique $\mathbf{a}$ et le potentiel scalaire électrique φ tels que $\mathbf{e} = -(\partial_t \mathbf{a} + grad\,\varphi)$. Nous obtenons alors :

$$\mathbf{rot}\,(\nu\,\mathbf{rot}\,\mathbf{a}) + \sigma\,(\partial_t \mathbf{a} + grad\,\varphi) = 0 \tag{1.2.27}$$

Les propriétés du potentiel vecteur magnétique impliquent la conservation de la composante normale de l'induction magnétique. Dans ces conditions, nous avons également conservation de la composante tangentielle du champ électrique. Enfin, l'équation 1.2.27 impose la conservation de la composante normale de la densité de courant et de la composante tangentielle du champ magnétique.

Formulation en potentiels t-Ω

Si nous prenons $\mathbf{h} = \mathbf{t} - grad\,\Omega$, nous obtenons à partir de l'équation 1.2.22 :

$$\mathbf{rot}\,(\eta\,\mathbf{rot}\,\mathbf{t}) + \partial_t \mu(\mathbf{t} - grad\,\Omega) = 0 \tag{1.2.28}$$

De même que précédemment, les propriétés de la densité de courant impliquent la continuité de la composante tangentielle du potentiel vecteur électrique et donc la conservation de la composante tangentielle du champ magnétique. Enfin, l'équation 1.2.28 impose la conservation de la composante normale de l'induction magnétique et de la composante tangentielle du champ électrique.

1.3 Complexe de De Rham

Du fait de la complexité géométrique des dipositifs traités, il nous est impossible de résoudre analytiquement les équations de Maxwell. Nous devons alors recourir à des méthodes numériques basées sur des techniques de discrétisation. La discrétisation exige une certaine "souplesse" de la solution. Or, les formulations de la section 1.2 sont exprimées sous forme forte, forme qui requiert une certaine régularité de la solution. Pour contourner ce problème, nous devons utiliser la forme faible (ou intégrale) de ces formulations. Il s'agit d'une forme plus "laxiste" que la forme forte mais qui présente l'avantage d'être plus facilement discrétisable. Le passage à la forme faible consiste à transformer le problème initial en un problème dit variationnel, ce dernier utilisant des espaces fonctionnels pour accueillir les champs et les potentiels qui leur sont associés. Avec les opérateurs différentiels gradient, rotationnel et divergence, ces espaces doivent former une structure mathématique continue permettant d'exprimer rigoureusement les équations de Maxwell. Une telle structure est encore appelée complexe de De Rham [9, 48].

Dans cette section, nous commençons par rappeler ce que sont les équations aux dérivées partielles et quelles sont les trois grandes classes d'équations possibles. Ce rappel nous permettra, par la suite, de classer les formulations de la section 1.2 et de justifier le choix de telle ou telle méthode de discrétisation. Nous présentons ensuite les outils permettant de passer au problème variationnel et le

processus de construction d'une structure mathématique continue. Nous présentons également le diagramme de Tonti qui permet de rattacher cette structure à sa structure duale.

Par la suite, nous considérerons les équations de Maxwell sur un domaine Ω, connexe, ouvert et borné de $\mathbb{R}^3$, de frontière $\partial\Omega$ notée Γ. Cette frontière se décompose en deux parties complémentaires Γ_h et Γ_b qui désignent respectivement des conditions à la limite sur le champ magnétique et sur l'induction magnétique. Le domaine Ω est muni d'un repère cartésien. La normale à la frontière Γ est orientée de l'intérieur du domaine vers l'extérieur. La figure 1.1 présente ce domaine d'étude.

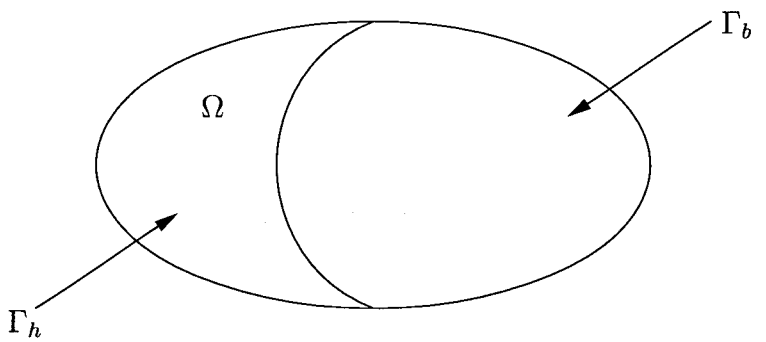

Fig. 1.1 – Domaine d'étude

1.3.1 Équations aux dérivées partielles

Une équation aux dérivées partielles est une relation entre une fonction et ses dérivées [46]. Si cette relation est linéaire par rapport à chacun de ses arguments, alors l'équation est dite linéaire. Elle est dite non-linéaire dans le cas contraire. Si ses coefficients sont constants, alors l'équation est dite à coefficients constants. Elle est dite de degré N si les dérivées qu'elle fait intervenir sont de degrés n, $0 \leq n \leq N$. Le domaine sur lequel est prise cette équation s'appelle le domaine de l'équation. Si la fonction inconnue est scalaire, l'équation est dite scalaire. Dans le cas contraire, nous parlons de système d'équations aux dérivées partielles scalaires.

Classes d'équations

Les équations aux dérivées partielles elliptiques, paraboliques et hyperboliques forment les trois grandes classes d'équations aux dérivées partielles [59]. Considérons à présent les équations aux dérivées partielles suivantes :

– équation elliptique ($u : \mathbb{R}^d \to \mathbb{R}$) :

$$-\triangle u = -\sum_{i=1}^{d} \partial^2_{x_i^2} u = f \tag{1.3.1}$$

- équation parabolique ($u : \mathbb{R}_+ \times \mathbb{R}^d \to \mathbb{R}$) :

$$\partial_t u - \triangle u = f \tag{1.3.2}$$

- équation hyperbolique ($u : \mathbb{R} \times \mathbb{R}^d \to \mathbb{R}$) :

$$\partial_{t^2}^2 u - \triangle u = f \tag{1.3.3}$$

Soit un point $\mathbf{y} = (y_1, ..., y_d) \in \mathbb{R}^d$ et soient $\widehat{u}$ et $\widetilde{u}$ les transformées de Fourier et de Laplace.

L'application de la transformée de Fourier à l'équation 1.3.1 conduit à :

$$(\sum_{i=1}^{d} y_i^2)\, \widehat{u}(\mathbf{y}) = \widehat{f} \tag{1.3.4}$$

Les points $\mathbf{y}$ vérifiant l'équation $|\mathbf{y}|^2 = 1$ forment une ellipse. C'est pourquoi nous parlons d'équation aux dérivées partielles elliptique.

La double application des transformées de Fourier (pour les variables d'espace) et de Laplace (pour la variable temps) à l'équation 1.3.2, conduit à :

$$(\tau - |\mathbf{y}|^2)\, \widetilde{u}(\mathbf{y}) = \widetilde{f} \tag{1.3.5}$$

Les points $\mathbf{y}$ vérifiant l'équation $\tau - |\mathbf{y}|^2 = 1$ forment une parabole. C'est pourquoi nous parlons d'équation aux dérivées partielles parabolique.

L'application de la transformée de Fourier (pour les variables d'espace et de temps) à l'équation 1.3.3, conduit à :

$$(-\tau^2 + |\mathbf{y}|^2)\, \widehat{u}(\mathbf{y}) = \widehat{f} \tag{1.3.6}$$

Les points $\mathbf{y}$ vérifiant l'équation $-\tau^2 + |\mathbf{y}|^2 = 1$ forment une hyperbole. C'est pourquoi nous parlons d'équation aux dérivées partielles hyperbolique.

Conditions aux limites

Pour qu'une équation aux dérivées partielles soit bien posée, il est nécessaire d'ajouter une information concernant la fonction inconnue sur toute ou partie de la frontière du domaine d'étude. Cette information est appelée condition à la limite. Elle peut être de type Dirichlet (valeur de la fonction), Neumann (valeur de la dérivée normale de la fonction), Robin (condition mixte Dirichlet-Neumann), ou plus complexe (condition artificielle) [7].

1.3.2 Formulation faible du problème

Un problème aux dérivées partielles bien posé est de la forme :

$$L\, u(x) = f(x), \quad \forall\, x \in \Omega \tag{1.3.7}$$

$$B\, u(x) = g(x), \quad \forall\, x \in \Gamma \tag{1.3.8}$$

où L est un opérateur différentiel d'ordre n, B est un opérateur définissant une condition à la limite, f et g sont deux fonctions respectivement définies sur le

domaine et sa frontière et u est une fonction inconnue appartenant à un espace fonctionnel V défini sur le domaine et sa frontière. Ce problème est alors exprimé sous forme forte.

Le passage à la forme faible (ou forme intégrale) consiste à transformer le problème initial en un problème dit variationnel. Les outils nécessaires à cette transformation sont les formules de Green. Pour $u \in dom(L)$ et $v \in dom(L^*)$, la formule de Green généralisée est définie par :

$$(L\,u, v) \; - \; (u, L^*\,v) = \int_\Gamma Q(u, v)\, ds \tag{1.3.9}$$

où L est un opérateur différentiel d'ordre n, L^* son opérateur adjoint d'ordre n et Q est une forme bilinéaire en u et v. Lorsque $L = L^*$, l'opérateur L est dit auto-adjoint. L'espace $dom(L^*)$ est appelé espace dual de $dom(L)$. La formule de Green généralisée est plus connue sous le nom de formule d'intégration par parties.

Problème variationnel

Soit $v \in V$ une fonction quelconque. L'intégration de l'équation 1.3.7 sur le domaine Ω, après multiplication par la fonction v, conduit à :

$$(L\,u, v) = (f, v), \quad u \in V \tag{1.3.10}$$

Cette intégration est rendue possible par le choix d'un espace fonctionnel V adéquat. L'application de la formule de Green généralisée fait apparaître une intégrale sur la frontière du domaine d'étude. Dès lors, la condition à la limite 1.3.8 peut être prise en compte et nous obtenons :

$$(u, L^*\,v) \; + \; \int_\Gamma Q_g(v)\, ds = (f, v), \quad u \in V \tag{1.3.11}$$

où Q_g est une forme linéaire en v, dépendante de g.

La fonction v est appelée fonction test. La relation 1.3.11 devant être vérifiée pour toutes les fonctions tests de l'espace fonctionnel V, nous obtenons le problème variationnel :

$$(u, L^*\,v) \; + \; \int_\Gamma Q_g(v)\, ds = (f, v), \quad \forall\, v \in V, u \in V \tag{1.3.12}$$

Le choix d'un espace fonctionnel V adéquat et d'une condition à la limite adéquate, permettent de dire que toute solution du problème variationnel est solution du problème initial. Les deux problèmes sont alors dits équivalents.

Nous remarquons que la régularité de la solution u exigée dans le problème initial, est reportée sur la fonction test dans le problème variationnel. Ce dernier est donc plus "laxiste" que le problème initial, c'est pourquoi nous parlons de formulation faible du problème initial.

1.3.3 Structure mathématique continue

Pour résoudre les équations de Maxwell dans le domaine Ω en utilisant la forme faible des formulations de la section 1.2, nous devons définir des espaces fonctionnels pour accueillir les champs et les potentiels qui leur sont associés [31]. Nous définissons ensuite des sous espaces qui permettent de prendre en compte les conditions aux limites sur Γ_h ou Γ_b. Ces sous-espaces, augmentés des opérateurs différentiels gradient, rotationnel et divergence, forment une structure mathématique continue qui permet d'exprimer rigoureusement les équations de Maxwell. Grâce aux opérateurs adjoints, nous pouvons lui associer une structure duale. Par la suite, nous considérerons que les conditions aux limites sur Γ_h sont associées à la structure primale et que les conditions à la limite sur Γ_b sont associées à la structure duale. Le diagramme de Tonti permet de relier ces structures par l'intermédiaire des lois de comportement.

Espaces fonctionnels

Soit L^2, l'espace des champs scalaires de carré intégrable sur Ω, auquel est associé le produit scalaire :

$$(u, v) = \int_\Omega u\, v\, d\Omega, \quad u, v \in L^2 \tag{1.3.13}$$

La norme correspondante est :

$$\| u \|_{L^2} = \sqrt{(u, u)}, \quad u \in L^2 \tag{1.3.14}$$

Soit $\mathbf{L^2}$, l'espace des champs vectoriels dont le carré de la norme Euclidienne est intégrable sur Ω et auquel est associé le produit scalaire :

$$(\mathbf{u}, \mathbf{v}) = \int_\Omega \mathbf{u} \odot \mathbf{v}\, d\Omega, \quad \mathbf{u}, \mathbf{v} \in \mathbf{L^2} \tag{1.3.15}$$

La norme correspondante est :

$$\| \mathbf{u} \|_{\mathbf{L^2}} = \sqrt{(\mathbf{u}, \mathbf{u})}, \quad \mathbf{u} \in \mathbf{L^2} \tag{1.3.16}$$

Les espaces L^2 et $\mathbf{L^2}$ sont des espaces de Sobolev qui nous permettent de construire les domaines de définition des opérateurs différentiels gradient, rotationnel et divergence [31, 48] :

$$E^0 = \{f \in L^2;\ grad\, f \in \mathbf{L^2}\} \tag{1.3.17}$$

$$\mathbf{E^1} = \{\mathbf{v} \in \mathbf{L^2};\ \mathbf{rot}\, \mathbf{v} \in \mathbf{L^2}\} \tag{1.3.18}$$

$$\mathbf{E^2} = \{\mathbf{u} \in \mathbf{L^2};\ \mathbf{div}\, \mathbf{u} \in \mathbf{L^2}\} \tag{1.3.19}$$

Structure primale et duale

Pour prendre en compte les conditions aux limites sur Γ_h, nous définissons quatre sous-espaces de degré $p = 0, 1, 2, 3$ et notés E_h^p. Ces sous-espaces, augmentés des opérateurs différentiels gradient, rotationnel et divergence, forment la structure primale suivante :

Degré	Opérateur	Domaine de définition
0	$grad_h$	$E_h^0 = \{f \in L^2;\ grad\, f \in \mathbf{L^2},\ f_{\vert\Gamma_h} = 0\}$
1	$\mathbf{rot_h}$	$\mathbf{E_h^1} = \{\mathbf{h} \in \mathbf{L^2};\ \mathbf{rot\ h} \in \mathbf{L^2},\ \mathbf{n} \otimes \mathbf{h}_{\vert\Gamma_h} = 0\}$
2	$\mathbf{div_h}$	$\mathbf{E_h^2} = \{\mathbf{j} \in \mathbf{L^2};\ \mathbf{div\ j} \in \mathbf{L^2},\ \mathbf{n} \odot \mathbf{j}_{\vert\Gamma_h} = 0\}$
3	0	$E_h^3 = \{u \in L^2\}$

Les domaines de définition des opérateurs sont construits de telle sorte que nous ayons les inclusions :

$$grad_h\, E_h^0 \subset \mathbf{E_h^1},\quad \mathbf{rot_h}\, \mathbf{E_h^1} \subset \mathbf{E_h^2},\quad \mathbf{div_h}\, \mathbf{E_h^2} \subset E_h^3 \tag{1.3.20}$$

et les égalités :

$$grad\, E_h^0 = ker(\mathbf{rot_h}),\quad \mathbf{rot_h}\, \mathbf{E_h^1} = ker(\mathbf{div_h}) \tag{1.3.21}$$

dans le domaine connexe Ω.

Ces égalités signifient que dans un domaine connexe, l'image du sous-espace E_h^0 par l'opérateur gradient est égal au noyau de l'opérateur rotationnel dans le sous-espace $\mathbf{E_h^1}$ et que l'image du sous-espace $\mathbf{E_h^1}$ par l'opérateur rotationnel est égale au noyau de l'opérateur divergence dans le sous-espace $\mathbf{E_h^2}$. Nous retrouvons ainsi les relations classiques $\mathbf{rot}(grad\, f) = 0$ et $\mathbf{div(rot\ u)} = 0$.

Ces égalités sont représentées dans le diagramme de De Rham pour le cas connexe (figure 1.2). Ce diagramme permet de mettre en évidence le fait que les opérateurs différentiels gradient, rotationnel et divergence, définissent des applications des espaces E_h^p vers les espaces E_h^{p+1}. Il permet également de mettre en évidence une décomposition orthogonale de l'espace $\mathbf{L^2}$.

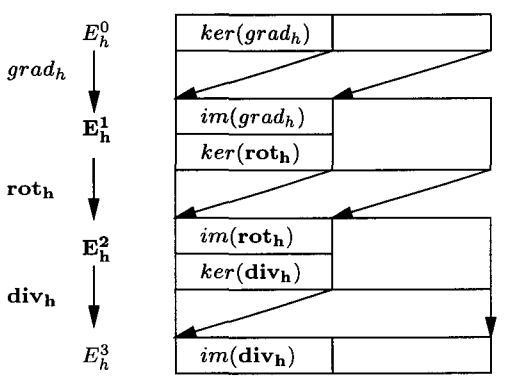

Fig. 1.2 – Diagramme de De Rham, cas connexe

La structure mathématique duale est obtenue en appliquant les formules de Green de type $grad - grad$ et **rot-rot**. Cette structure permet de prendre en compte les conditions aux limites sur Γ_b. Elle est donnée par :

Degré	Opérateur	Domaine de définition
3	0	$E_b^3 = \{u \in L^2\}$
2	$(grad_h)^* = -\,\mathbf{div_b}$	$\mathbf{E_b^2} = \{\mathbf{b} \in \mathbf{L^2};\ \mathbf{div\ b} \in \mathbf{L^2},\ \mathbf{n} \odot \mathbf{b}_{\vert\Gamma_b} = 0\}$
1	$(\mathbf{rot_h})^* = \mathbf{rot_b}$	$\mathbf{E_b^1} = \{\mathbf{h} \in \mathbf{L^2};\ \mathbf{rot\ h} \in \mathbf{L^2},\ \mathbf{n} \otimes \mathbf{h}_{\vert\Gamma_b} = 0\}$
0	$(\mathbf{div_h})^* = -\,grad_b$	$E_b^0 = \{f \in L^2;\ grad\, f \in \mathbf{L^2},\ f_{\vert\Gamma_b} = 0\}$

où la notation $*$ désigne l'opérateur adjoint.

Comme pour la structure primale, nous avons les inclusions :

$$grad_b\, E_b^0 \subset \mathbf{E_b^1}, \quad \mathbf{rot_b}\, \mathbf{E_b^1} \subset \mathbf{E_b^2}, \quad \mathbf{div_b}\, \mathbf{E_b^2} \subset E_b^3 \tag{1.3.22}$$

et les égalités :

$$grad_b\, E_b^0 = ker(\mathbf{rot_b}), \quad \mathbf{rot_b}\, \mathbf{E_b^1} = ker(\mathbf{div_b}) \tag{1.3.23}$$

dans le domaine connexe Ω.

Ces structures permettent d'exprimer rigoureusement les équations de Maxwell. Ainsi, dans le cas des formulations magnétostatiques de la section 1.2, nous constatons que le potentiel scalaire magnétique Ω, le champ source $\mathbf{h_s}$ et le potentiel vecteur magnétique $\mathbf{a}$ appartiennent respectivement aux espaces E_h^0, $\mathbf{E_h^1}$ et $\mathbf{E_b^1}$. Dans le cas des formulations magnétodynamiques de cette même section, nous constatons que le potentiel scalaire magnétique Ω, le potentiel vecteur électrique $\mathbf{t}$, le potentiel scalaire électrique φ et le potentiel vecteur magnétique $\mathbf{a}$ appartiennent respectivement aux espaces E_h^0, $\mathbf{E_h^1}$, E_b^0 et $\mathbf{E_b^1}$. Quant à la densité de courant $\mathbf{j}$ et l'induction magnétique $\mathbf{b}$, celles-ci appartiennent respectivement aux espaces $\mathbf{E_h^2}$ et $\mathbf{E_b^2}$.

Le diagramme de Tonti permet de relier la structure primale et la structure duale par l'intermédaire des lois de comportement. La figure 1.3 présente ce diagramme dans le cas magnétodynamique [9, 31, 48].

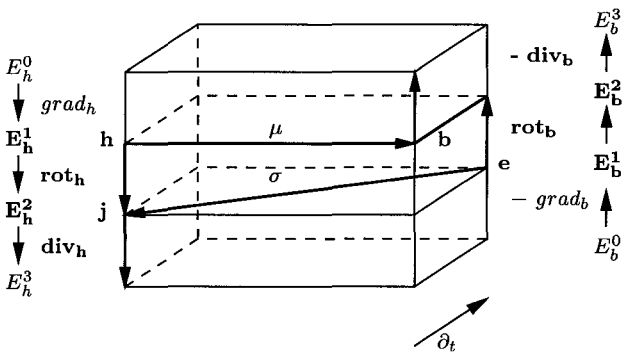

Fig. 1.3 – Diagramme de Tonti, cas magnétodynamique

1.4 Complexe de Whitney

Pour résoudre les équations de Maxwell, nous devons maintenant remplacer le problème continu par un problème discret analogue mais caractérisé par un nombre fini d'inconnues. Pour ce faire, nous devons remplacer les espaces fonctionnels de la section 1.3 par des espaces discrets encore appelés espaces d'approximation. Ces derniers, augmentés des équivalents discrets des opérateurs différentiels gradient, rotationnel et divergence, doivent constituer une structure mathématique discrète capable d'exprimer les équations de Maxwell de manière aussi rigoureuse que la structure mathématique continue.

Les méthodes de différences finies, éléments finis et volumes finis sont des méthodes de discrétisation. La méthode des différences finies peut s'employer pour toutes les classes d'équations aux dérivées partielles. Cependant, elle ne permet pas de traiter des géométries complexes [59], c'est pourquoi son utilisation est souvent restreinte à la discrétisation temporelle dans les problèmes évolutifs. La méthode des éléments finis est principalement utilisée pour les équations aux dérivées partielles elliptiques et paraboliques. Contrairement à la méthode des différences finies, elle est très bien adaptée aux géométries complexes. Les formulations magnétostatiques et magnétodynamiques de la section 1.2 étant respectivement des équations elliptiques et paraboliques, il est logique que nous utilisions cette méthode. La méthode des volumes finis est la plus récente de toutes. C'est une méthode dédiée aux problèmes hyperboliques et pour laquelle des outils d'estimation d'erreur sont actuellement en cours d'étude [28].

Les espaces d'éléments finis mixtes de Nédélec ont marqué l'introduction de la géométrie différentielle dans les techniques de discrétisation. En montrant que le système de Maxwell était lié au formalisme des formes différentielles, A. Bossavit [9] a démontré que les éléments finis mixtes de Nédélec étaient en fait des formes différentielles discrètes : les éléments finis de Whitney [13]. Les espaces d'approximation relatifs aux éléments finis de Whitney constituent une suite d'espaces analogue à la suite d'espaces fonctionnels continus de la section 1.3. Augmentés des équivalents discrets des opérateurs différentiels gradient, rotationnel et divergence, ils constituent la structure mathématique discrète que nous recherchons pour exprimer rigoureusement les équations de Maxwell. Une telle structure est encore appelée complexe de Whitney.

Dans cette section, nous commençons par présenter la méthode des éléments finis afin d'introduire plus facilement les éléments finis de Whitney. Nous en profitons pour présenter la méthode de Galerkin, méthode bien connue et permettant d'obtenir des systèmes d'équations algébriques symétriques et définis positifs. Nous présentons ensuite le processus de construction de la structure mathématique discrète.

Par la suite, nous supposerons un maillage M_h du domaine d'étude Ω, composé de tétraèdres à quatre nœuds, de prismes à six nœuds et de cubes à huit nœuds. Nous désignerons respectivement par N, A, F et V les ensembles de nœuds, d'arêtes, de facettes et de volumes de ce maillage. Leurs cardinaux respectifs seront notés $card(N)$, $card(A)$, $card(F)$ et $card(V)$.

1.4.1 Méthode des éléments finis

Discrétiser un domaine d'étude consiste à le mailler. Un maillage est un assemblage d'éléments géométriques de formes simples. La méthode des éléments finis est basée sur la double discrétisation des espaces fonctionnels continus et du domaine d'étude. Sur chaque élément géométrique, les inconnues du problème sont discrétisées par une combinaison de fonctions de base spatiales continues. Ces fonctions peuvent porter sur les nœuds, les arêtes, les facettes ou le volume de l'élément. La continuité des fonctions de base permet d'interpoler la valeur de la fonction inconnue en tout point du domaine étudié.

Elément fini

Un élément fini est défini par un triplet (E, W_E, X_E) tel que [31] :

- E est un élément géométrique de forme simple;
- W_E est l'espace des fonctions de base spatiales définies sur E. Il s'agit d'un espace fonctionnel de dimension finie n_E noté $\{w_i, 1 \leq i \leq n_E\}$;
- X_E est un ensemble de fonctionnelles linéaires $\{x_i, 1 \leq i \leq n_E\}$, définies sur W_E et à valeur dans $\mathbb{R}$.

Les fonctions de l'espace W_E sont parfois appelées fonctions de forme. Elles permettent d'interpoler la valeur d'une fonction en tout point de E et d'assurer des conditions de continuité à la traversée des interfaces entre E et ses voisins.

Les fonctionnelles de l'espace X_E sont appelées degrés de liberté. Pour illustrer cette notion, nous allons interpoler une fonction u, supposée suffisamment régulière, dans W_E. Une telle interpolation s'écrit sous la forme d'une combinaison linéaire des fonctions de W_E :

$$u_E = \sum_{i=1}^{n_E} a_i \, w_i, \quad u_E \in W_E \tag{1.4.1}$$

La propriété d'unisolvance de l'élément fini garantit l'unicité de cette interpolation et permet d'écrire :

$$x_i(u - u_E) = 0, \quad \forall \, x_i \in X_E, u_E \in W_E \tag{1.4.2}$$

ce qui conduit au système d'équations linéaires suivant :

$$x_j(u) = \sum_{i=1}^{n_E} a_i \, x_j(w_i), \quad 1 \leq i, j \leq n_E \tag{1.4.3}$$

Choisissons à présent des fonctionnelles x_i telles que :

$$x_j(w_i) = \delta_{ij}, \quad 1 \leq i, j \leq n_E \tag{1.4.4}$$

où δ_{ij} est le symbole de Kronecker. Nous pouvons alors écrire :

$$a_i = x_i(u), \quad 1 \leq i \leq n_E \tag{1.4.5}$$

Dès lors, l'interpolation de la fonction u dans l'espace W_E peut s'écrire sous la forme :

$$u_E = \sum_{i=1}^{n_E} x_i(u) \, w_i, \quad u_E \in W_E \tag{1.4.6}$$

Les coefficients $x_i(u)$ sont appelés degrés de liberté. De manière informelle, ce sont les inconnues du problème relatives à l'élément fini.

Espaces d'éléments finis

Nous avons vu précédemment que les fonctions de l'espace W_E permettaient d'interpoler la valeur d'une fonction en tout point de E et d'assurer des conditions de continuité à la traversée des interfaces entre E et ses voisins. Il est alors possible d'interpoler la valeur de la fonction en tout point du domaine d'étude en construisant un espace d'éléments finis.

Considérons le maillage M_h du domaine d'étude Ω. Nous associons un élément fini (E, W_E, X_E) à chaque élément géométrique E de ce maillage. Nous pouvons alors construire un espace d'éléments finis (M_h, W_h, X_h) tel que :

- M_h est le maillage constitué de tous les éléments géométriques E;
- W_h est l'espace des fonctions de base construit à partir des espaces W_E. Il s'agit d'un espace de dimension finie n_h noté $\{w_{h,i}, 1 \leq i \leq n_h\}$;
- X_h est un ensemble de fonctionnelles linéaires $\{x_{h,i}, 1 \leq i \leq n_h\}$ qui sont liées aux fonctionnelles des espaces X_E.

Les fonctionnelles de X_h vérifient la condition :

$$x_{h,j}(w_{h,i}) = \delta_{ij}, \quad 1 \leq i, j \leq n_h \tag{1.4.7}$$

Dès lors, l'interpolée u_h d'une fonction u suffisament régulière est définie de manière unique par :

$$u_h = \sum_{i=1}^{n_h} x_{h,i}(u)\, w_{h,i}, \quad u_h \in W_h \tag{1.4.8}$$

et vérifie :

- la restriction $u_h|_E$ appartient à l'espace W_E. Elle est entièrement déterminée par la donnée de l'ensemble $X_E(u)$ des degrés de liberté de u;
- des conditions de continuité sont assurées à la traversée des interfaces entre éléments géométriques.

Méthode de Galerkin

Reprenons le problème variationnel exprimé par l'équation 1.3.12 :

$$(u, L^* v) + \int_\Gamma Q_g(v)\, ds = (f, v), \quad \forall\, v \in V, u \in V \tag{1.4.9}$$

et tentons d'y introduire l'interpolation 1.4.8. Il est clair que dans le cas général, nous ne pourrons jamais satisfaire cette équation en prenant des fonctions tests quelconques. Par contre, si nous choisissons ces fonctions tests de telle sorte que :

$$v = v_i, \quad 1 \leq i \leq n_h \tag{1.4.10}$$

l'équation 1.4.9 se transforme en un système d'équations algébriques :

$$(u_h, L^* v_j) + \int_\Gamma Q_g(v_j)\, ds = (f, v_j), \quad 1 \leq j \leq n_h \tag{1.4.11}$$

La méthode de Galerkin consiste alors à choisir les fonctions tests v_j parmi les fonctions de base de l'espace W_h. Cette méthode conduit fréquemment à l'obtention de systèmes d'équations symétriques et définis positifs [67].

Notons que si E désigne le ième élément de M_h et si Ω^E et Γ^E désignent respectivement les parties de Ω et Γ occupées par E, nous pouvons écrire :

$$(u_h, L^* v_j) = \sum_{E=1}^{card(V)} (u_h, L^* v_j)_{\Omega^E}, \quad 1 \leq j \leq n_h \tag{1.4.12}$$

$$(f, v_j) = \sum_{E=1}^{card(V)} (f, v_j)_{\Omega^E}, \quad 1 \leq j \leq n_h \tag{1.4.13}$$

sur le domaine Ω et :

$$\int_\Gamma Q_g(v_j)\, ds = \sum_{E=1}^{card(V)} \int_{\Gamma^E} Q_g(v_j)\, ds, \quad 1 \leq j \leq n_h \tag{1.4.14}$$

sur la frontière Γ.

Dès lors, le système d'équations précédent peut être construit élément par élément en calculant la contribution de chaque élément et en sommant toutes ces contributions pour obtenir le système final. Cette construction est encore appelée phase d'assemblage du système d'équations.

1.4.2 Structure mathématique discrète

Les éléments finis de Whitney sont une famille d'éléments finis relatifs aux nœuds, aux arêtes, aux facettes et au volume d'un même élément géométrique. Nous parlons alors d'éléments finis nodaux, d'arêtes, de facettes et de volumes. Les espaces d'éléments finis qu'ils engendrent permettent de remplacer les espaces fonctionnels continus de la section 1.3 et ainsi d'accueillir les champs et les potentiels qui leur sont associés au niveau discret. Les matrices d'incidence sont les équivalents discrets des opérateurs différentiels gradient, rotationel et divergence. Avec les espaces d'éléments finis nodaux, d'arêtes, de facettes et de volumes, elles forment une structure mathématique discrète analogue à la structure mathématique continue de la section 1.3.

Par la suite, nous emploierons les notations proposées par P. Dular dans [31]. De plus, afin de faciliter la lecture, les grandeurs continues et leurs formes discrétisées seront représentées par les mêmes notations.

Espaces d'approximation

Pour un élément géométrique E de M_h, nous désignons par :

- n_i un nœud du maillage appartenant à E;
- a_{ij} l'arête de E reliant les nœuds n_i et n_j;
- f une facette de E définie par une liste circulaire de $card(f)$ nœuds. Les nœuds q_{c-1} et q_{c+1} sont respectivement le prédécesseur et le successeur du nœud q_c dans cette liste;

- v le volume de E;
- a_f un entier positif valant 2 pour une facette à 3 nœuds et 1 pour une facette à 4 nœuds;
- $N_{F,i\bar{j}}$ l'ensemble des nœuds de la facette f, adjacente à l'arête a_{ij}, qui contient le nœud n_i, mais pas le nœud n_j.

Soit $\lambda(x, y, z)$ un point de E et $p_i(\lambda)$ une fonction relative au nœud n_i. Cette fonction prend la valeur 1 sur ce nœud et s'annule sur les autres nœuds du maillage. Elle varie continûment entre les éléments géométriques qui partagent le nœud n_i. Par la suite, cette fonction sera simplement notée p_i.

Considérons à présent les fonctions de base suivantes :

- au nœud n_i est associée la fonction :

$$w_{n_i} = p_i \tag{1.4.15}$$

- à l'arête a_{ij} est associée la fonction :

$$\mathbf{w_{a_{ij}}} = p_j \, grad\Big(\sum_{r \in N_{F,j\bar{i}}} p_r\Big) - p_i \, grad\Big(\sum_{r \in N_{F,i\bar{j}}} p_r\Big) \tag{1.4.16}$$

- à la facette f est associée la fonction :

$$\mathbf{w_f} = a_f \sum_{c=1}^{card(f)} \Big(p_{q_c} \, grad\Big(\sum_{r \in N_{F,q_c\bar{q}_{c+1}}} p_r\Big) \otimes grad\Big(\sum_{r \in N_{F,q_c\bar{q}_{c-1}}} p_r\Big)\Big) \tag{1.4.17}$$

- au volume v est associée la fonction :

$$w_v = \frac{1}{v} \tag{1.4.18}$$

Ces fonctions possèdent les propriétés suivantes :

- la fonction w_n vaut 1 sur le nœud n et 0 sur les autres nœuds. Nous avons alors :

$$w_i(n_j) = \delta_{ij}, \quad \forall\,(i, j) \in N \times N \tag{1.4.19}$$

  Cette fonction est continue à travers les facettes;
- La circulation de $\mathbf{w_a}$ vaut 1 le long de l'arête a et 0 le long des autres arêtes. Nous avons alors :

$$\int_j \mathbf{w_i} \odot \mathbf{dl} = \delta_{ij}, \quad \forall\,(i, j) \in A \times A \tag{1.4.20}$$

  La composante tangentielle de $\mathbf{w_a}$ est continue à travers les facettes;
- Le flux de $\mathbf{w_f}$ vaut 1 à travers la facette f et 0 à travers les autres facettes. Nous avons alors :

$$\int_j \mathbf{w_i} \odot \mathbf{n} \, ds = \delta_{ij}, \quad \forall\,(i, j) \in F \times F \tag{1.4.21}$$

  La composante normale de $\mathbf{w_f}$ est continue à travers les facettes;

- L'intégrale volumique de w_v vaut 1 sur le volume v et 0 sur les autres volumes. Nous avons alors :

$$\int_j w_i \, dv = \delta_{ij}, \quad \forall \, (i, j) \in V \times V \tag{1.4.22}$$

Cette fonction est discontinue.

Nous constatons que les degrés de liberté associés aux fonctions de base w_n, $\mathbf{w_a}$, $\mathbf{w_f}$ et w_v sont respectivement des évaluations ponctuelles, des intégrales curvilignes, surfaciques et volumiques. Les éléments finis associés à ces fonctions sont respectivement les éléments finis nodaux, d'arêtes, de facettes et de volumes. Soient W^0, $\mathbf{W^1}$, $\mathbf{W^2}$ et W^3 les espaces d'approximation respectivement construits sur ces éléments finis.

Les conditions de transmission à l'interface entre deux milieux de propriétés différentes, imposent la continuité des composantes tangentielles des champs $\mathbf{h}$ et $\mathbf{e}$, ainsi que celle des potentiels vecteurs $\mathbf{a}$ et $\mathbf{t}$. Elles imposent également la continuité des composantes normales de $\mathbf{b}$ et $\mathbf{j}$. De par les propriétés de continuité des fonctions de base aux interfaces entre éléments géométriques, l'espace $\mathbf{W^1}$ permet d'exprimer $\mathbf{h}$, $\mathbf{e}$, $\mathbf{a}$ et $\mathbf{t}$, tandis que l'espace $\mathbf{W^2}$ permet d'exprimer $\mathbf{b}$ et $\mathbf{j}$. Les espaces W^0 et W^3 permettent d'exprimer respectivement les potentiels scalaires φ et la densité volumique de charges ρ.

Matrices d'incidence

La notion d'incidence est définie par :

- l'incidence d'un nœud n dans une arête a, notée $i(n, a)$, vaut 1 si n est l'extrémité de a, -1 si n est l'origine de a et 0 si n n'appartient pas à a;
- l'incidence d'une arête a dans une facette f, notée $i(a, f)$, vaut 1 si la liste ordonnée des nœuds de a est une sous-liste directe dans la liste circulaire des nœuds de f, -1 si c'est une sous-liste indirecte et 0 si f ne contient pas a;
- l'incidence d'une facette f dans un volume v, notée $i(f, v)$, vaut 1 si la normale à f, déterminée par la règle de la main droite en parcourant ses nœuds, est extérieure à v, -1 si elle est intérieure à v, et 0 si f n'appartient pas à v.

et conduit aux relations [31] :

$$grad \, w_n = \sum_{a \in A} i(n, a) \, \mathbf{w_a} \tag{1.4.23}$$

$$\mathbf{rot} \, \mathbf{w_a} = \sum_{f \in F} i(a, f) \, \mathbf{w_f} \tag{1.4.24}$$

$$\mathbf{div} \, \mathbf{w_f} = \sum_{v \in V} i(f, v) \, w_v \tag{1.4.25}$$

Les matrices d'incidence G_{AN}, $\mathbf{R_{FA}}$ et $\mathbf{D_{VF}}$ sont des matrices rectangulaires, respectivement définies par :

$$G_{a,n} = i(n,a), \quad \forall\, a \in A, \forall\, n \in N \tag{1.4.26}$$
$$R_{f,a} = i(a,f), \quad \forall\, f \in F, \forall\, a \in A \tag{1.4.27}$$
$$D_{v,f} = i(f,v), \quad \forall\, v \in V, \forall\, f \in F \tag{1.4.28}$$

Soit Ω un potentiel scalaire magnétique. Nous pouvons écrire :

$$\Omega = \sum_{n \in N} \Omega_n \, w_n, \quad \Omega \in W^0 \tag{1.4.29}$$

En introduisant cette expression dans la relation 1.4.23, nous obtenons :

$$\mathbf{h} = -\, grad\, \Omega = -\sum_{a \in A} \Big(\sum_{n \in N} i(n,a)\, \Omega_n \Big)\, \mathbf{w_a}, \quad \Omega \in W^0 \tag{1.4.30}$$

Le champ magnétique $\mathbf{h}$ étant un élément de $\mathbf{W^1}$, nous pouvons écrire :

$$\mathbf{h} = \sum_{a \in A} h_a \, \mathbf{w_a}, \quad \mathbf{h} \in \mathbf{W^1} \tag{1.4.31}$$

Si (h_a) et (Ω_n) désignent respectivement les vecteurs colonnes des composantes de $\mathbf{h}$ et Ω, nous avons :

$$(h_a) = -\, G_{AN}\, (\Omega_n) \tag{1.4.32}$$

Dès lors, la matrice d'incidence G_{AN} est l'équivalent discret de l'opérateur différentiel gradient.

En introduisant l'expression du champ magnétique 1.4.31 dans la relation 1.4.24, nous obtenons :

$$\mathbf{j} = \mathbf{rot}\, \mathbf{h} = \sum_{f \in F} \Big(\sum_{a \in A} i(a,f)\, h_a \Big)\, \mathbf{w_f}, \quad \mathbf{h} \in \mathbf{W^1} \tag{1.4.33}$$

La densité de courant $\mathbf{j}$ étant un élément de $\mathbf{W^2}$, nous pouvons écrire :

$$\mathbf{j} = \sum_{f \in F} j_f \, \mathbf{w_f}, \quad \mathbf{j} \in \mathbf{W^2} \tag{1.4.34}$$

Si (j_f) désigne le vecteur colonne des composantes de $\mathbf{j}$, nous avons :

$$(j_f) = \mathbf{R_{FA}}\, (h_a) \tag{1.4.35}$$

Dès lors, la matrice d'incidence $\mathbf{R_{FA}}$ est l'équivalent discret de l'opérateur différentiel rotationnel. La relation $\mathbf{rot}\ (grad\, \Omega) = 0$ est vérifiée au niveau discret par $\mathbf{R_{FA}}\, G_{AN} = 0$.

Considérons à présent la divergence de la densité de courant. Nous pouvons écrire :

$$\mathbf{div}\,\mathbf{j} = \sum_{v \in V} \Big(\sum_{f \in F} i(f,v)\, j_f \Big)\, w_v, \quad \mathbf{j} \in \mathbf{W^2} \tag{1.4.36}$$

De par la définition de la densité de courant, la somme des flux de facettes sur un élément est nulle (loi de conservation du flux). Nous avons ainsi :

$$(0) = \mathbf{D_{VF}}\ (j_f) \tag{1.4.37}$$

Dès lors, la matrice d'incidence $\mathbf{D_{VF}}$ est l'équivalent discret de l'opérateur différentiel divergence. La relation $\mathbf{div\ (rot\ h)} = 0$ est vérifiée au niveau discret par $\mathbf{D_{VF}}\ \mathbf{R_{FA}} = 0$.

Structures

Les espaces d'approximation W^0, $\mathbf{W^1}$, $\mathbf{W^2}$ et W^3 sont les domaines de définition des matrices d'incidence. Comme pour la structure continue, nous pouvons définir des sous-espaces pour prendre en compte les conditions aux limites sur Γ_h et Γ_b. Nous pouvons ainsi construire la structure suivante :

Degré	Opérateur	Domaine de définition
0	$grad_h$	$W_h^0 = \{f \in W^0;\ grad\, f \in \mathbf{W^1},\ f\vert_{\Gamma_h} = 0\}$
1	$\mathbf{rot_h}$	$\mathbf{W_h^1} = \{\mathbf{h} \in \mathbf{W^1};\ \mathbf{rot\ h} \in \mathbf{W^2},\ \mathbf{n} \otimes \mathbf{h}\vert_{\Gamma_h} = 0\}$
2	$\mathbf{div_h}$	$\mathbf{W_h^2} = \{\mathbf{j} \in \mathbf{W^2};\ \mathbf{div\ j} \in W^3,\ \mathbf{n} \odot \mathbf{j}\vert_{\Gamma_h} = 0\}$
3	0	$W_h^3 = \{u \in W^3\}$

Comme pour la structure mathématique continue, les domaines de définition des opérateurs sont construits de telle sorte que que nous ayons les inclusions :

$$grad_h\, W_h^0 \subset \mathbf{W_h^1}, \quad \mathbf{rot_h}\, \mathbf{W_h^1} \subset \mathbf{W_h^2}, \quad \mathbf{div_h}\, \mathbf{W_h^2} \subset W_h^3 \tag{1.4.38}$$

et les égalités :

$$grad\, W_h^0 = ker(\mathbf{rot_h}), \quad \mathbf{rot_h}\, \mathbf{W_h^1} = ker(\mathbf{div_h}) \tag{1.4.39}$$

dans le domaine connexe Ω.

Ces égalités signifient que dans un domaine connexe, l'image du sous-espace W_h^0 par la matrice d'incidence G_{AN} est égal au noyau de la matrice d'incidence $\mathbf{R_{FA}}$ dans le sous-espace $\mathbf{W_h^1}$ et que l'image du sous-espace $\mathbf{W_h^1}$ par la matrice d'incidence $\mathbf{R_{FA}}$ est égale au noyau de la matrice d'incidence $\mathbf{D_{VF}}$ dans le sous-espace $\mathbf{W_h^2}$. Ces égalités sont représentées dans le diagramme de De Rham discret de la figure 1.4.

De manière analogue à la structure mathématique continue, nous pouvons construire une structure permettant de prendre en compte les conditions à la limite sur Γ_b :

Degré	Opérateur	Domaine de définition
3	0	$W_b^3 = \{u \in W^3\}$
2	$(grad_h)^* = -\,\mathbf{div_b}$	$\mathbf{W_b^2} = \{\mathbf{b} \in \mathbf{W^2};\ \mathbf{div\ b} \in W^3,\ \mathbf{n} \odot \mathbf{b}\vert_{\Gamma_b} = 0\}$
1	$\mathbf{(rot_h)^* = rot_b}$	$\mathbf{W_b^1} = \{\mathbf{a} \in \mathbf{W^1};\ \mathbf{rot\ a} \in \mathbf{W^2},\ \mathbf{n} \otimes \mathbf{a}\vert_{\Gamma_b} = 0\}$
0	$\mathbf{(div_h)^*} = -\,grad_b$	$W_b^0 = \{f \in W^0;\ grad\, f \in \mathbf{W^1},\ f\vert_{\Gamma_b} = 0\}$

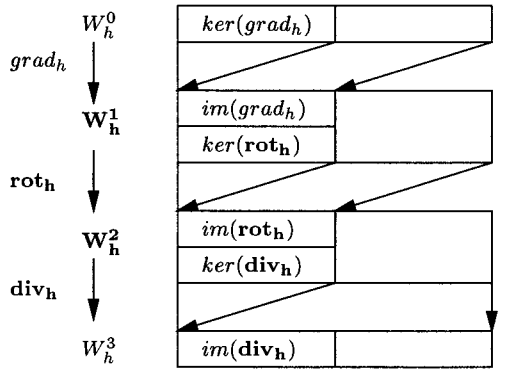

Fig. 1.4 – Diagramme de De Rham, cas connexe

De même que pour la structure précédente, nous avons les inclusions :

$$grad_b\, W_b^0 \subset \mathbf{W_b^1}, \quad \mathbf{rot_b}\, \mathbf{W_b^1} \subset \mathbf{W_b^2}, \quad \mathbf{div_b}\, \mathbf{W_b^2} \subset W_b^3 \tag{1.4.40}$$

et les égalités dans le domaine connexe Ω :

$$grad_b\, W_b^0 = ker(\mathbf{rot_b}), \quad \mathbf{rot_b}\, \mathbf{W_b^1} = ker(\mathbf{div_b}) \tag{1.4.41}$$

Comme dans le cas continu, ces deux structures sont reliées par le diagramme de Tonti discret de la figure 1.5.

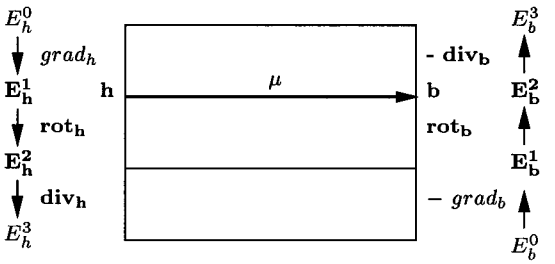

Fig. 1.5 – Diagramme de Tonti, cas magnétostatique

1.5 Discrétisation des formulations

Dans cette section, nous commençons par présenter les résultats de travaux de Ren [54] sur les problèmes auto-jaugés. Nous développons ensuite les formes discrétisées des formulations de la section 1.2 [6, 10]. Pour se conformer aux travaux de Ren, ces formulations ne sont pas jaugées.

Afin de ne pas introduire de confusion entre les notations désignant le domaine d'étude et certains potentiels, nous considérons un domaine d'étude D de frontière Γ, analogue à celui de la section 1.3. D_c et Γ_c désignent respectivement le domaine conducteur et sa frontière. D_0 désigne le domaine inducteur. Les fonctions tests associées à une entité α sont notées α'. Dans un couplage

entre formulations magnétostatiques et magnétodynamiques, la normale à la frontière est orientée de D_c vers $D - D_c$. Le maillage considéré est celui de la section 1.4.

La méthode de Galerkin est utilisée pour la discrétisation spatiale. La formule implicite d'Euler est utilisée pour la discrétisation temporelle. La dérivée temporelle d'une fonction u est donnée par :

$$\partial_t u = \frac{u_{t+\Delta t} - u_t}{\Delta t}, \quad t_0 \leq t \leq t_f \; et \; u_{t_0} = u_0 \tag{1.5.1}$$

avec :

t_0	l'instant initial
t_f	l'instant final
u_0	la valeur initiale de u
Δt	le pas de temps, variable ou constant

Par la suite, les intégrales de surface seront notées :

$$< u, v >_\Gamma \; = \; \int_\Gamma u \, v \, d\Gamma, \quad u, v \in L^2 \tag{1.5.2}$$

$$< \mathbf{u}, \mathbf{v} >_\Gamma \; = \; \int_\Gamma \mathbf{u} \odot \mathbf{v} \, d\Gamma, \quad \mathbf{u}, \mathbf{v} \in \mathbf{L^2} \tag{1.5.3}$$

1.5.1 Problèmes auto-jaugés

La section 1.2 a présenté des jauges possibles pour garantir l'unicité de la solution quand des potentiels vecteurs électriques ou magnétiques sont employés. Ces jauges peuvent être la jauge de Coulomb $\mathbf{div\,a} = 0$ ou bien encore la jauge $\mathbf{a} \odot \mathbf{w} = 0$ dans laquelle $\mathbf{w}$ est un champ vectoriel dont les lignes ne se referment pas [2]. Au niveau continu, ces jauges doivent être ajoutées en tant que termes supplémentaires dans les équations aux dérivées partielles. Aprés discrétisation des formulations, il en résulte un surcroît de calcul durant les phases d'assemblage et de résolution des systèmes d'équations correspondants.

Une formulation compatible est une formulation dont tous les termes sont discrétisés sur les éléments finis de Whitney. Récemment, Ren a montré que la jauge est inutile lorsqu'une formulation compatible et la méthode du gradient conjugué sont employées conjointement, le problème étant alors auto-jaugé. Ce résultat est dû au fait que la méthode du gradient conjugué utilise implicitement une forme faible de la jauge de Coulomb et qu'elle la maintient constante tout au long de ses itérations.

Pour illustrer ce résultat, considérons les formulations de la section 1.2. Ces formulations utilisent des termes $grad\,\Omega$ et $\mathbf{rot\,a}$ où Ω et $\mathbf{a}$ désignent respectivement un potentiel scalaire et un potentiel vecteur. Nous supposerons que ces formulations ne sont pas jaugées. Le terme gradient est défini à une constante près. En général, des conditions aux limites sont imposées sur Ω. Ces conditions permettent de fixer la constante et font ainsi office de jauge. En l'absence de conditions aux limites, il existe une infinité de potentiels Ω. Le terme $\mathbf{rot\,a}$ étant défini à un gradient près, il existe systématiquement une infinité de potentiels vecteurs $\mathbf{a}$.

Les matrices des systèmes d'équations que nous obtenons sont symétriques. Lorsque les formulations sont jaugées, elles sont définies positives. En l'absence de jauge, ces matrices sont dites surgénératives puisqu'il y a plus d'équations que d'inconnues. Elles sont donc symétriques et semi-définies positives. De telles matrices ne peuvent être directement traitées par des méthodes directes, mais peuvent l'être par la méthode du gradient conjugué qui travaille implicitement dans l'image de la matrice [46].

1.5.2 Formulations magnétostatiques

Nous cherchons à calculer le champ magnétique dans le domaine d'étude D. Celui ci peut contenir des matériaux magnétiques non-linéaires ($\mu = \mu(\mathbf{h})$). Les sources du champ proviennent du domaine inducteur D_0 où la densité de courant $\mathbf{j_0}$ imposée est connue.

Formulation en potentiel scalaire magnétique

La forme faible de cette formulation est donnée par :

$$(\mu\ grad\ \Omega', grad\ \Omega)_D + < \Omega'\ \mathbf{b}, \mathbf{n} >_\Gamma = (\mu\ \mathbf{h_s}, grad\ \Omega')_D \tag{1.5.4}$$

Le potentiel scalaire magnétique Ω est discrétisé dans l'espace W^0 des éléments finis nodaux. La continuité de la composante tangentielle du champ magnétique est ainsi assurée de manière forte. La continuité de la composante normale de l'induction magnétique étant implicite, nous pouvons annuler le terme surfacique de l'équation 1.5.4 : la continuité de cette composante est ainsi assurée de manière faible. La forme discrétisée de l'équation 1.5.4 est alors :

$$\sum_{i=1}^{card(N)} (\mu\ grad\ w_j, grad\ w_i)_D\ \Omega_i = (\mu\ \mathbf{h_s}, grad\ w_j)_D, \quad 1 \leq j \leq card(N) \tag{1.5.5}$$

Le sytème 1.5.5 s'écrit sous la forme matricielle :

$$W\ X = F \tag{1.5.6}$$

avec :

$$\begin{cases} W_{ji} = (\mu\ grad\ w_j, grad\ w_i)_D, & 1 \leq i, j \leq card(N) \\ F_j = (\mu\ \mathbf{h_s}, grad\ w_j)_D, & 1 \leq j \leq card(N) \end{cases}$$

Pour obtenir une formulation compatible, le champ source $\mathbf{h_s}$ doit être discrétisé dans l'espace $\mathbf{W}^1$ des éléments finis d'arêtes. Une telle discrétisation peut être obtenue par la technique de l'arbre d'arêtes [48] qui est basée sur le théorème d'Ampère. La matrice W est creuse, symétrique, définie ou semi-définie positive. Le vecteur X contient la valeur du potentiel scalaire magnétique Ω sur les nœuds du maillage.

Formulation en potentiel vecteur a

La forme faible de cette formulation est donnée par :

$$(\nu \, \mathbf{rot}\, \mathbf{a}', \mathbf{rot}\, \mathbf{a})_D - < \mathbf{a}', \mathbf{h} \otimes \mathbf{n} >_\Gamma = (\mathbf{a}', \mathbf{j_0})_{D_0} \tag{1.5.7}$$

Le potentiel vecteur magnétique $\mathbf{a}$ est discrétisé dans l'espace $\mathbf{W^1}$ des éléments finis d'arêtes. La continuité de la composante normale de l'induction magnétique est ainsi assurée de manière forte. La continuité de la composante tangentielle du champ magnétique étant implicite, nous pouvons annuler le terme surfacique de l'équation 1.5.7 : la continuité de cette composante est ainsi assurée de manière faible. La forme discrétisée de l'équation 1.5.7 est alors :

$$\sum_{i=1}^{card(A)} (\nu \, \mathbf{rot}\, \mathbf{w_j}, \mathbf{rot}\, \mathbf{w_i})_D \, C_{\mathbf{a},i} = (\mathbf{w_j}, \mathbf{j_0})_{D_0}, \quad 1 \leq j \leq card(A) \tag{1.5.8}$$

Le système 1.5.8 s'écrit sous la forme matricielle :

$$W \, X = F \tag{1.5.9}$$

avec :

$$\begin{cases} W_{ji} = (\nu \, \mathbf{rot}\, \mathbf{w_j}, \mathbf{rot}\, \mathbf{w_i})_D, & 1 \leq i, j \leq card(A) \\ F_j = (\mathbf{w_j}, \mathbf{j_0})_{D_0}, & 1 \leq j \leq card(A) \end{cases}$$

Pour obtenir une formulation compatible, la densité de courant $\mathbf{j_0}$ doit être discrétisée dans l'espace $\mathbf{W^2}$ des éléments finis de facettes et être à divergence nulle. Une telle discrétisation peut être obtenue par la technique de l'arbre de facettes [48] qui utilise la loi de conservation du flux dans l'élément. La matrice W est creuse, symétrique et semi-définie positive. Le vecteur X contient la circulation du potentiel vecteur magnétique $\mathbf{a}$ sur les arêtes du maillage.

1.5.3 Formulations magnétodynamiques

Nous cherchons à calculer la distribution du champ magnétique et des courants induits dans le domaine d'étude D. Celui-ci contient des matériaux conducteurs et des matériaux magnétiques éventuellement non-linéaires. Les inducteurs sont caractérisés par une densité de courant imposée connue. Un couplage entre formulations magnétodynamiques et magnétostatiques permet d'assurer les conditions de continuité entre régions conductrices et non-conductrices.

Formulation en potentiels a-φ

Dans le domaine conducteur, la forme faible de cette formulation est donnée par :

$$(\nu \, \mathbf{rot}\, \mathbf{a}', \mathbf{rot}\, \mathbf{a})_{D_c} + (\sigma \, \mathbf{a}', \partial_t \mathbf{a} + grad\, \varphi)_{D_c} - < \mathbf{a}', \mathbf{h} \otimes \mathbf{n} >_{\Gamma_c} = 0 \tag{1.5.10}$$

et par :

$$(\sigma \, grad\, \varphi', \partial_t \mathbf{a} + grad\, \varphi)_{D_c} + < \varphi', \mathbf{j} \odot \mathbf{n} >_{\Gamma_c} = 0 \tag{1.5.11}$$

La continuité de la composante normale de la densité de courant et la continuité de la composante tangentielle du champ magnétique étant implicites, nous pouvons annuler les termes surfaciques des équations 1.5.10 et 1.5.11 : la continuité de ces composantes est ainsi assurée de manière faible.

Dans le domaine non-conducteur, nous utilisons la formulation en potentiel vecteur magnétique **a**. Nous trouvons ainsi le potentiel vecteur magnétique **a** sur tout le domaine D, le potentiel scalaire électrique φ sur le domaine conducteur D_c et la densité de courant sur le domaine inducteur D_0.

Le potentiel scalaire électrique φ est discrétisé dans l'espace W^0 des éléments finis nodaux. Le potentiel vecteur magnétique **a** est discrétisé dans l'espace $\mathbf{W^1}$ des éléments finis d'arêtes. La continuité de la composante tangentielle du champ électrique et la continuité de la composante normale de l'induction magnétique sont ainsi assurées de manière forte.

Nous obtenons alors les nouvelles équations :

$$(\nu\ \mathbf{rot}\ \mathbf{a}', \mathbf{rot}\ \mathbf{a})_D\ +\ (\sigma\ \mathbf{a}', \partial_t \mathbf{a}\ +\ grad\ \varphi)_{D_c} = (\mathbf{a}', \mathbf{j_0})_{D_0} \tag{1.5.12}$$

et :

$$(\sigma\ grad\ \varphi', \partial_t \mathbf{a}\ +\ grad\ \varphi)_{D_c} = 0 \tag{1.5.13}$$

Les formes discrétisées de ces équations sont :

$$\sum_{i=1}^{card(A)} ((\nu\ \mathbf{rot}\ \mathbf{w_j}, \mathbf{rot}\ \mathbf{w_i})_D\ C_{\mathbf{a},i}\ +\ (\sigma\ \mathbf{w_j}, \mathbf{w_i})_{D_c}\ \partial_t C_{\mathbf{a},i})\ +$$

$$\sum_{k=1}^{card(N)} (\sigma\ \mathbf{w_j}, grad\ w_k)_{D_c}\ \varphi_k = (\mathbf{w_j}, \mathbf{j_0})_{D_0}, \quad 1 \leq j \leq card(A) \tag{1.5.14}$$

et :

$$\sum_{i=1}^{card(A)} (\sigma\ \mathbf{w_i}, grad\ w_j)_{D_c}\ \partial_t C_{\mathbf{a},i}\ +$$

$$\sum_{k=1}^{card(N)} (\sigma\ grad\ w_j, grad\ w_k)_{D_c}\ \varphi_k = 0, \quad 1 \leq j \leq card(N) \tag{1.5.15}$$

Ce système s'écrit sous la forme matricielle :

$$\begin{pmatrix} W_{\mathbf{a}} + T_{\mathbf{a}}/\Delta t & U_{\mathbf{a}-\varphi} \\ U^T_{\mathbf{a}-\varphi} & \Delta t\, W_\varphi \end{pmatrix} \begin{pmatrix} X_1 \\ X_2 \end{pmatrix}_{t+\Delta t} =$$

$$\begin{pmatrix} F_{\mathbf{a}} \\ 0 \end{pmatrix}_{t+\Delta t} + \begin{pmatrix} T_{\mathbf{a}}/\Delta t & 0 \\ U^T_{\mathbf{a}-\varphi} & 0 \end{pmatrix} \begin{pmatrix} X_1 \\ X_2 \end{pmatrix}_t \tag{1.5.16}$$

avec :

$$\begin{cases} W_{\mathbf{a},ji} = (\nu\ \mathbf{rot}\ \mathbf{w_j}, \mathbf{rot}\ \mathbf{w_i})_D, & 1 \leq i,j \leq card(A) \\ T_{\mathbf{a},ji} = (\sigma\ \mathbf{w_j}, \mathbf{w_i})_{D_c}, & 1 \leq i,j \leq card(A) \\ U_{\mathbf{a}-\varphi,ji} = (\sigma\ \mathbf{w_j}, grad\ w_i)_{D_c}, & 1 \leq i \leq card(N), 1 \leq j \leq card(A) \\ W_{\varphi,ji} = (\sigma\ grad\ w_j, grad\ w_i)_{D_c}, & 1 \leq i,j \leq card(N) \\ F_{\mathbf{a},j} = (\mathbf{w_j}, \mathbf{j_0})_{D_0}, & 1 \leq j \leq card(A) \end{cases}$$

Pour obtenir une formulation compatible, la densité de courant $\mathbf{j_0}$ doit être discrétisée dans l'espace $\mathbf{W}^2$ des éléments finis de facettes et être à divergence nulle. La matrice du système est creuse, symétrique et semi-définie positive. Les sous-vecteurs X_1 et X_2 contiennent respectivement la circulation du potentiel vecteur magnétique $\mathbf{a}$ sur les arêtes du maillage et la valeur du potentiel scalaire électrique φ sur les nœuds de ce maillage.

Formulation en potentiels t-Ω

Dans le domaine conducteur, la forme faible de cette formulation est donnée par :

$$(\frac{1}{\sigma}\ \mathbf{rot}\ \mathbf{t}', \mathbf{rot}\ \mathbf{t})_{D_c} + (\mathbf{t}', \partial_t \mu(\mathbf{t} - grad\ \Omega))_{D_c} - < \mathbf{t}', \mathbf{e} \otimes \mathbf{n} >_{\Gamma_c} = 0 \tag{1.5.17}$$

et par :

$$-(\mu\ grad\ \Omega', \mathbf{t} - grad\ \Omega)_{D_c} + < \Omega'\ \mathbf{b}, \mathbf{n} >_{\Gamma_c} = 0 \tag{1.5.18}$$

La continuité de la composante tangentielle du champ électrique et la continuité de la composante normale de l'induction magnétique étant implicites, nous pouvons annuler les termes surfaciques des équations 1.5.17 et 1.5.18 : ces continuités sont ainsi assurées de manière faible.

Dans le domaine non-conducteur, nous utilisons la formulation en potentiel scalaire magnétique. Nous trouvons ainsi le potentiel scalaire magnétique Ω et le champ source $\mathbf{h_s}$ sur tout le domaine D et le potentiel vecteur électrique $\mathbf{t}$ sur le domaine conducteur D_c.

L'annulation de la composante tangentielle du potentiel vecteur électrique $\mathbf{t}$ assure de manière forte la continuité de la composante normale de la densité de courant, ainsi que la continuité de la composante tangentielle du champ magnétique, le potentiel scalaire magnétique Ω étant choisi continu.

Nous obtenons alors les nouvelles équations :

$$(\frac{1}{\sigma}\ \mathbf{rot}\ \mathbf{t}', \mathbf{rot}\ \mathbf{t})_{D_c} + (\mathbf{t}', \partial_t \mu(\mathbf{t} - grad\ \Omega))_{D_c} = 0 \tag{1.5.19}$$

et :

$$-(\mu\ grad\ \Omega', \mathbf{t})_{D_c} + (\mu\ grad\ \Omega', grad\ \Omega)_D = (\mu\ \mathbf{h_s}, grad\ \Omega')_D \tag{1.5.20}$$

Le potentiel scalaire magnétique Ω est discrétisé dans l'espace W^0 des éléments finis nodaux. Le potentiel vecteur électrique $\mathbf{t}$ est discrétisé dans l'espace

$\mathbf{W^1}$ des éléments finis d'arêtes. Les formes discrétisées des équations 1.5.19 et 1.5.20 sont alors :

$$\sum_{i=1}^{card(A)} ((\frac{1}{\sigma}\ \mathbf{rot\ w_j},\ \mathbf{rot\ w_i})_{D_c}\ C_{\mathbf{t},i}\ +\ (\mu\ \mathbf{w_j},\ \mathbf{w_i})_{D_c}\ \partial_t C_{\mathbf{t},i})\ -$$

$$\sum_{k=1}^{card(N)} (\mu\ \mathbf{w_j},\ grad\ w_k)_{D_c}\ \Omega_k = 0, \quad 1 \leq j \leq card(A) \tag{1.5.21}$$

et :

$$-\ \sum_{i=1}^{card(A)} (\mu\ \mathbf{w_i},\ grad\ w_j)_{D_c}\ C_{\mathbf{t},i}\ +\ \sum_{k=1}^{card(N)} (\mu\ grad\ w_j,\ grad\ w_k)_D\ \Omega_k =$$

$$(\mu\ \mathbf{h_s},\ grad\ w_j)_D, \quad 1 \leq j \leq card(N) \tag{1.5.22}$$

Ce système s'écrit sous la forme matricielle :

$$\begin{pmatrix} \Delta t\, W_{\mathbf{t}} + T_{\mathbf{t}} & U_{\mathbf{t}-\Omega} \\ U^T_{\mathbf{t}-\Omega} & W_\Omega \end{pmatrix} \begin{pmatrix} X_{\mathbf{t}} \\ X_\Omega \end{pmatrix}_{t+\Delta t} =$$

$$\begin{pmatrix} 0 \\ F_\Omega \end{pmatrix}_{t+\Delta t} + \begin{pmatrix} T_{\mathbf{t}} & U_{\mathbf{t}-\Omega} \\ 0 & 0 \end{pmatrix} \begin{pmatrix} X_{\mathbf{t}} \\ X_\Omega \end{pmatrix}_t \tag{1.5.23}$$

avec :

$$\begin{cases} W_{\mathbf{t},ji} &= (\frac{1}{\sigma}\ \mathbf{rot\ w_j},\ \mathbf{rot\ w_i})_{D_c}, & 1 \leq i,j \leq card(A) \\ T_{\mathbf{t},ji} &= (\mu\ \mathbf{w_j},\ \mathbf{w_i})_{D_c}, & 1 \leq i,j \leq card(A) \\ U_{\mathbf{t}-\Omega,ji} &= -\ (\mu\ \mathbf{w_j},\ grad\ w_i)_{D_c}, & 1 \leq i \leq card(N), 1 \leq j \leq card(A) \\ W_{\Omega,ji} &= (\mu\ grad\ w_j,\ grad\ w_i)_D, & 1 \leq i,j \leq card(N) \\ F_{\Omega,j} &= (\mu\ \mathbf{h_s},\ grad\ w_j)_D, & 1 \leq j \leq card(N) \end{cases}$$

Pour obtenir une formulation compatible, comme dans le cas magnétostatique, le champ source $\mathbf{h_s}$ doit être discrétisé dans l'espace $\mathbf{W^1}$ des éléments finis d'arêtes. La matrice du système est creuse, symétrique définie ou semi-définie positive. Les sous-vecteurs X_t et X_Ω contiennent respectivement la circulation du potentiel vecteur électrique $\mathbf{t}$ sur les arêtes du maillage et la valeur du potentiel scalaire magnétique Ω sur les nœuds de ce maillage.

1.6 Conclusion

La section 1.2 a présenté les formulations utilisées par les codes numériques pour résoudre les équations de la magnétostatique et de la magnétodyamique.

Les formulations magnétostatiques en potentiel scalaire sont intéressantes dans le cas des domaines simplement connexes, en raison du faible nombre d'inconnues utilisé. Leur exploitation se révèle plus délicate dans le cas des domaines multiplement connexes. Dans ce cas, il faut soit introduire des coupures artificielles, soit mailler les vides pour retrouver un domaine simplement connexe. La formulation en potentiel vecteur magnétique est applicable dans les deux types de domaines. Sa mise en œuvre est cependant plus lourde en raison du nombre d'inconnues plus élevé.

Les formulations magnétodynamiques hybrides mettant en jeu un potentiel vecteur et un potentiel scalaire, respectivement discrétisés dans les espaces d'éléments finis nodaux et d'arêtes, permettent de surmonter les problèmes de discontinuités à l'interface entre deux milieux de propriétés différentes, mais le nombre d'inconnues utilisé est relativement important.

De nombreux problèmes physiques, une fois transformés en problèmes variationnels, se posent en termes de fonctions inconnues, scalaires ou vectorielles, sur les espaces fonctionnels E^0, $\mathbf{E^1}$ et $\mathbf{E^2}$. Des conditions aux limites homogènes sur la frontière du domaine d'étude permettent de fixer les variables dans dans E^0, les composantes tangentielles dans $\mathbf{E^1}$ et les composantes normales dans $\mathbf{E^2}$. Ces espaces sont donc d'une grande importance dans le domaine de la physique puisque $\mathbf{E^1}$ est l'espace idéal pour représenter les champs électriques et magnétiques, tandis que $\mathbf{E^2}$ est l'espace idéal pour représenter les quantités qui obéissent à une loi de conservation du flux. Ils permettent d'obtenir la forme faible du problème initial, qui est beaucoup plus facile à discrétiser que la forme forte. Ces espaces forment, avec les opérateurs différentiels gradient, rotationnel et divergence, une structure mathématique continue que nous avons présentée dans la section 1.3. Cette structure permet d'exprimer rigoureusement les équations de Maxwell.

Le principe d'une méthode de discrétisation est de transformer un problème continu en un problème discret analogue mais caractérisé par un nombre fini d'inconnues. Les formulations magnétostatiques et magnétodynamiques de la section 1.2 sont respectivement des équations aux dérivées partielles elliptiques et paraboliques. La méthode des éléments finis est alors la technique de discrétisation appropriée. Les espaces d'éléments finis nodaux, d'arêtes, de facettes et de volumes permettent de conserver les propriétés du niveau continu au niveau discret. Dès lors, ils forment avec les équivalents discrets des opérateurs différentiels gradient, rotationnel et divergence, une structure mathématique discrète analogue à la structure mathématique continue. Cette structure a été présentée dans la section 1.4.

La section 1.5 a présenté les formes discrétisées des formulations de la section 1.2. Les matrices des systèmes d'équations correspondants sont creuses, symétriques, définies ou semi-définies positives. Le caractère creux de ces matrices est le corollaire de la méthode des éléments finis. Le fait qu'elles soient symétriques et définies positives résulte de l'application de la méthode de Galerkin. La semi-positivité est liée à l'absence de jauge. Les travaux de Ren ont montré que cette jauge n'était pas nécessaire quand une formulation compatible et la méthode du gradient conjugué étaient employées conjointement.

De nombreux problèmes électromagnétiques peuvent être modélisés en deux

dimensions en privilégiant une direction pour le courant. Cependant, certains problèmes nécessitent un calcul en trois dimensions afin de tenir compte de la géométrie ou des effets d'extrémités. Du fait du grand nombre d'inconnues requis par le domaine tridimensionnel, de la non-linéarité et de l'évolutivité dans le temps, les exemples traités restent relativement modestes. En effet, malgré les performances des méthodes de résolution de systèmes d'équations employées, qu'elles soient directes ou itératives, les limites en capacité mémoire et en puissance de calcul des stations de travail sont rapidement atteintes. Dans ce contexte, l'utilisation de calculateurs parallèles permet de modéliser des exemples plus proches de la réalité. Les calculateurs parallèles et les différentes manières de les programmer font l'objet du chapitre suivant.

Chapitre 2

Parallélisme

2.1 Introduction

Deux approches sont possibles pour accroître la puissance de calcul des machines. La première, séquentielle, consiste à exécuter les instructions plus rapidement. La seconde, parallèle, consiste à exécuter simultanément plusieurs instructions. Du fait des limites physiques en matière d'intégration électronique, les gains obtenus par l'approche séquentielle sont bornés à terme [29]. L'approche parallèle apparaît alors comme une alternative à l'approche séquentielle.

Les recherches sur ces deux approches ont été menées conjointement. Si les limites physiques de l'approche séquentielle ne sont pas encore atteintes, l'approche parallèle a été généralisée. Aujourd'hui, la plupart des processeurs utilisent plusieurs pipelines d'instructions, plusieurs unités de contrôle (processeurs superscalaires), des jeux d'instructions vectorielles et un mode d'exécution dans le désordre dans lequel plusieurs instructions peuvent être exécutées simultanément (exécution spéculative). Les serveurs web ou de base de données sont souvent basés sur des machines bi ou quadri-processeurs. Enfin, les serveurs de calcul peuvent contenir plusieurs dizaines ou plusieurs centaines de processeurs.

Une machine parallèle contient des processeurs, de la mémoire et un réseau de communication reliant les processeurs à cette mémoire. Selon le type de la machine, la mémoire et le réseau de communication peuvent adopter des architectures bien différentes. Le fonctionnement interne de cette machine est régi par un modèle d'exécution qui décrit les relations entre les instructions et les données auxquelles elles s'appliquent. La section 2.2 présente les différents types de machines parallèles ainsi que les différents modèles d'exécution qui les régissent.

La programmation d'une machine parallèle est régie par un modèle de programmation étroitement lié à son modèle d'exécution. Ce modèle utilise des techniques et des outils plus ou moins complexes à maîtriser. La qualité d'une application parallèle est évaluée en fonction de trois critères appelés facteur d'accélération, facteur d'efficacité et facteur de scalabilité (terme francisé). Ces facteurs sont bornés par la loi d'Amdhal. La section 2.3 présente les différents modèles de programmation, les facteurs d'évaluation et la loi d'Amdhal.

2.2 Machines parallèles

Toute machine contenant plus d'un proceseur est une machine parallèle. De manière plus formelle, une machine parallèle peut être définie par un quadruplet (P, M, R, E) tel que :

- P est un ensemble de processeurs;
- M est une mémoire;
- R est un réseau de communication qui relie les processeurs à la mémoire;
- E est un modèle d'exécution qui décrit les relations entre les instructions et les données auxquelles elles s'appliquent.

Il y a encore quelques années, deux types bien distincts de machines parallèles se partageaient le marché : les machines parallèles à mémoire partagée et les machines parallèles à mémoire répartie. Aujourd'hui, nous assistons à l'émergence d'un nouveau type de machine parallèle, à mi-chemin entre les deux types cités précédemment : les machines parallèles à base de SMP (Symmetric Multi Processors).

Dans cette section, nous commençons par présenter ces trois types de machines parallèles. Nous présentons ensuite les différents modèles d'exécution possibles en nous basant sur la classification de Flynn [34]. Cette dernière, quoique maintenant ancienne, est toujours largement utilisée. Notons que certains classifications étendues plus récentes permettent de tenir compte de l'organisation des machines ainsi que de l'accès à leur mémoire [53].

2.2.1 Machines à mémoire partagée

Dans une machine à mémoire partagée, tous les processeurs accèdent à une mémoire commune via le réseau de communication. Pour permettre à plusieurs processeurs d'accéder simultanément à la mémoire, celle-ci est physiquement découpée en sous-espaces appelés bancs. Il peut y avoir autant d'accès simultanés qu'il y a de bancs. Cette découpe est transparente pour l'utilisateur. La figure 2.1 présente l'architecture générale d'une machine à mémoire partagée, dans laquelle les processeurs (p_i) accèdent aux bancs (m_j) de la mémoire via le réseau de communication.

La communication inter-processeurs s'effectue par l'intermédiaire de variables partagées dans la mémoire. Comme le réseau de communication fait transiter des données et des instructions, il doit être performant. Lorsque le nombre de bancs est peu important, il est possible de créer un réseau totalement connecté dans lequel chaque processeur est directement relié à tous les bancs de la mémoire. Lorsque le nombre de bancs est trop important, le réseau est bâti sur des étages de commutateurs qui ralentissent d'autant la communication entre le processeur et le banc.

Dans une machine mono-processeur, la mémoire cache est une petite mémoire située entre le processeur et la mémoire principale. Son rôle est de stocker les données et les instructions les plus couramment utilisées par un code en cours d'exécution sur le processeur. Son fonctionnement est directement géré par le matériel. Il est totalement invisible pour l'utilisateur. Le temps d'accès à cette mémoire cache étant très inférieur à celui de la mémoire principale, les

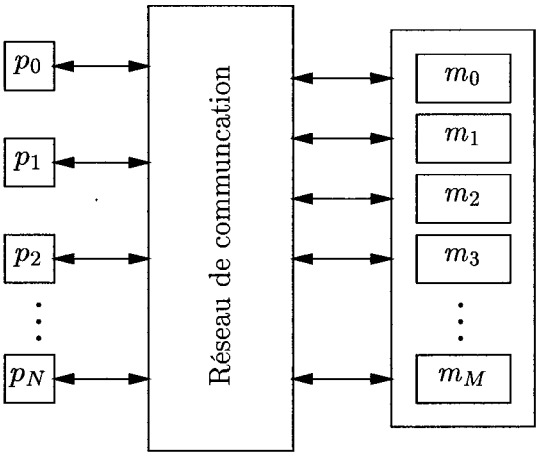

Fig. 2.1 – Machine à mémoire partagée

performances du code peuvent être améliorées de manière significative. L'utilisation des mémoires cache pose un problème dans le cas des machines parallèles à mémoire partagée. Dans ces machines, comme nous l'avons dit précédemment, les processeurs communiquent par l'intermédiaire de variables partagées dans la mémoire centrale. Chaque processeur qui communique dispose donc d'une copie de ces variables dans sa mémoire cache. Dès lors, toute modification de l'une de ces copies doit être répercutée au niveau de la mémoire centrale et au niveau de toutes les mémoires cache. Ceci suppose l'existence d'un mécanisme matériel permettant de garantir la cohérence des données. Ce mécanisme est basé sur un système d'invalidation des données périmées.

Le point fort de ces machines est la facilité d'utilisation d'une mémoire commune pouvant être accédée dans sa globalité par n'importe quel processeur. Le point faible est le petit nombre de processeurs qu'il est possible de connecter à la mémoire, du fait de la complexité et de la performance du réseau de communication sous-jacent, ainsi que de la complexité du mécanisme garantissant la cohérence des mémoires cache.

Parmi les machines parallèles à mémoire partagée les plus connues, citons le C90 de Cray. Notons qu'à l'heure actuelle, les machines à mémoire partagée sont de plus en plus remplacées par des machines SMP. Ces dernières sont également construites autour d'une mémoire partagée. La différence entre ces machines est à rechercher au niveau des temps d'accès à cette mémoire. Dans une machine SMP, le temps d'accès est rigoureusement le même pour tous les processeurs de la machine (d'où le nom de SMP). Dans une machine à mémoire partagée classique, ce temps d'accès est proportionnel au nombre de commutateurs disposés entre le processeur et le banc mémoire.

2.2.2 Machines à mémoire répartie

Dans une machine à mémoire répartie, chaque processeur dispose d'un espace mémoire local qu'il est seul à pouvoir adresser. L'espace mémoire global de

la machine est la juxtaposition de ces espaces locaux. La figure 2.2 présente l'architecture générale d'une machine à mémoire distribuée, dans laquelle chaque processeur p_i dispose de son propre espace mémoire local m_i et communique avec les autres processeurs via le réseau de communication.

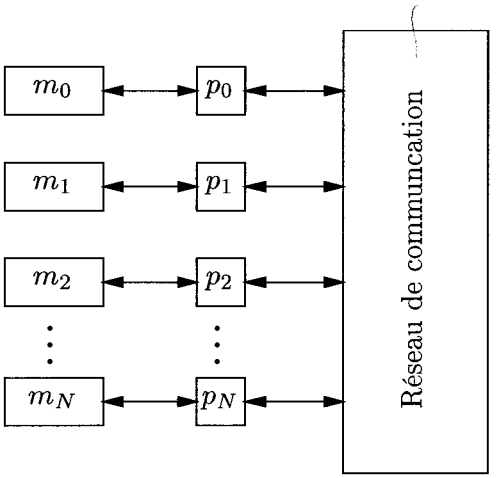

Fig. 2.2 – Machine à mémoire distribuée

La communication inter-processeurs s'effectue par l'intermédiaire de messages. Le corps d'un message ne contient que des données et son en-tête contient, entre autres, l'adresse du processeur cible. De fait, le réseau de communication d'une machine à mémoire répartie ne fait transiter que des données. Comme le nombre de processeurs d'une telle machine peut être important, il n'est pas possible de créer un réseau totalement connecté. Les formes les plus couramment rencontrées sont l'arbre, l'anneau, la grille et l'hypercube. Dans un tel réseau, l'acheminement d'un message est assuré par un algorithme de routage qui analyse l'en-tête de ce dernier et le dirige vers le processeur cible correspondant. Plus encore que pour une machine à mémoire partagée, en raison du grand nombre de processeurs qu'il est possible de connecter, le réseau de communication d'une machine à mémoire répartie doit être performant.

Le premier point fort de ces machines est le principe de répartition physique de la mémoire. Il permet aussi bien de créer des machines massivement parallèles avec beaucoup de processeurs, que de connecter des machines mono-processeur entre elles afin d'obtenir des machines parallèles à moindre coût. Ces dernières sont encore appelées fermes ou grappes de machines. Le second point fort est qu'elles ne nécessitent pas de mécanisme permettant de garantir la cohérence des mémoires cache, les communications inter-processeurs s'effectuant par l'intermédiaire de messages. Le premier point faible est la difficulté d'utilisation engendrée par le fractionnement physique de la mémoire. Le second point faible est la qualité du réseau de communication dans un contexte où la performance des processeurs croît beaucoup plus rapidement que celle des réseaux. Ainsi, les grappes de machines sont généralement connectées en anneau. Elles utilisent des réseaux à large bande passante (réseaux gigabits) tels que FastEthernet ou Myrinet. Cependant, ces réseaux sont caractérisés par une forte latence et un

surcoût logiciel important pour le traitement des messages. A l'inverse, dans les machines massivement parallèles, le traitement des messages est souvent assuré par le matériel via un processeur de communication associé à chaque processeur de calcul, ce qui permet d'éviter le surcoût logiciel propre aux grappes de machines. C'est notamment le cas de la CM5 de Thinking Machine Corporation avec son réseau de communication en hyperarbre. Cependant, la réalisation de tels réseaux est une tâche complexe.

Parmi les machines à mémoire répartie les plus connues, citons le T3E de Cray, la SP2 d'IBM et la CM5 de Thinking Machine Corporation. Ce type de machines a dominé le marché durant ces dernières années. Aujourd'hui, il est de plus en plus détrôné au profit des machines parallèles à base de SMP.

2.2.3 Machines à base de SMP

Les machines SMP (Symmetric Multi Processors) étaient à l'origine des machines à mémoire partagée dotées d'un petit nombre de processeurs (de 4 à 8) et du réseau de communication le plus simple qui soit : le bus. Aujourd'hui, ce bus est de plus en plus souvent remplacé par un crossbar ou un réseau d'alignement, ce qui permet d'intégrer un plus grand nombre de processeurs (de 16 à 32). La caractéristique principale de ces machines est de garantir un temps d'accès à la mémoire identique pour tous les processeurs.

Le réseau de communication des machines SMP ne permet pas de connecter un grand nombre de processeurs. Pour construire des machines plus importantes, il faut utiliser les machines SMP comme des briques de base et les interconnecter. Nous parlons alors de machines à base de SMP. L'Entreprise 10000 de Sun est un représentant bien connu de ce nouveau type de machines. Dans ces dernières, la brique de base est appelée noeud de calcul. La figure 2.3 présente l'architecture générale d'une machine à base de SMP.

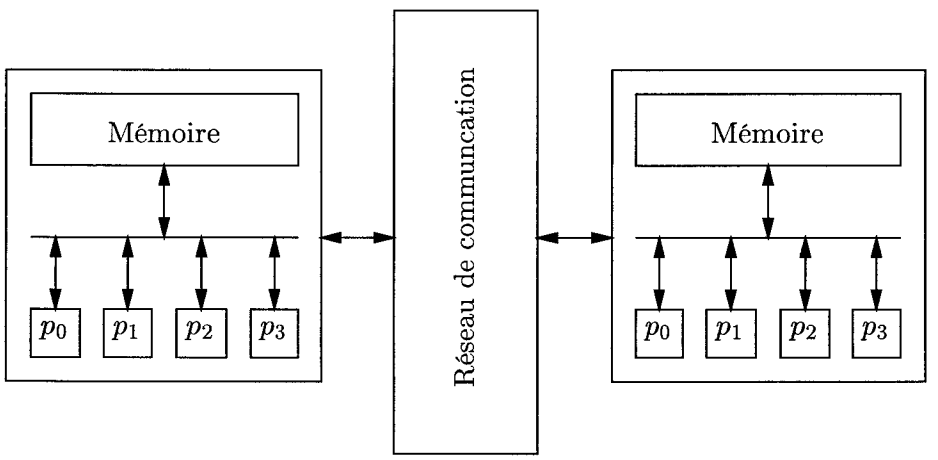

Fig. 2.3 – Machine à base de SMP

Comme pour les machines à mémoire répartie, il est possible de créer des machines massivement parallèles à base de SMP et des grappes de machines

SMP.

Dans le cas des machines massivement parallèles, le réseau de communication est interne et complètement dédié. Il peut intégrer un mécanisme d'adressage global permettant à chaque noeud de calcul d'accéder à la mémoire des autres noeuds. Ce mécanisme doit alors être complété par un autre mécanisme permettant de garantir la cohérence des mémoires cache. Nous parlons alors de machines à cohérence de cache et à temps d'accès mémoire non-uniforme (CC-NUMA). Ces dernières sont encore appelées machines à mémoire physiquement répartie et logiquement partagée. C'est notamment le cas de l'Origin 2000 de Silicon Graphics [43] dans laquelle le temps d'accès aux mémoires distantes est de deux à trois fois plus long que le temps d'accès à la mémoire locale. Avec une machine de type CC-NUMA, l'utilisateur retrouve le confort d'utilisation des machines à mémoire partagée, ce qui explique en partie le succès commercial de l'Origin 2000.

Les grappes de machines SMP utilisent une technologie de réseaux très récente : les réseaux à capacité d'adressage, conçus pour pallier les carences des réseaux gigabits classiques tels que FastEthernet ou Myrinet. Dans ces réseaux, le traitement des messages est assuré par le matériel (cartes) et n'implique pas le système d'exploitation. Le temps d'accès aux mémoire distantes est de l'ordre de la microseconde. Dès lors, les caractéristiques de ces réseaux sont des bandes passantes très larges et des latences très faibles. SCI (Scalable Coherent Interface) est le représentant le plus connu de cette nouvelle génération [37]. Des travaux de recherche visent à utiliser ces réseaux et à implanter des mécanismes permettant de garantir la cohérence des mémoires cache, afin de transformer une grappe de machines SMP en machine CC-NUMA [27].

2.2.4 Modèles d'exécution

Le modèle d'exécution est lié à l'architecture de la machine. Il décrit les relations entre les instructions et les données auxquelles elles s'appliquent. Parmi les classifications existantes, celle de Flynn [34] est la plus couramment utilisée. Cette classification repose sur l'étude des flux d'instructions et de données. Quoique maintenant ancienne, elle est toujours largement utilisée.

Un programme est constitué d'instructions et de données. Lors de son exécution, il est tout d'abord chargé en mémoire. La suite des instructions en mémoire est appelée flux d'instructions. La suite des données en mémoire auxquelles s'applique le flux d'instructions est appelée flux de données.

Dans une machine mono-processeur classique, chaque instruction est lue en mémoire par l'unité de contrôle qui l'envoie ensuite au processeur. Ce dernier applique l'instruction à la donnée correspondante puis stocke le résultat en mémoire. Comme une machine parallèle peut contenir plusieurs mémoires et plusieurs unités de contrôle, les flux d'instructions et de données peuvent être simples ou multiples. Flynn distingue les quatre grandes classes de modèles d'exécution de la figure 2.4.

		Flux de données	
		Unique	Multiple
Flux d'instructions	Unique	SISD	SIMD
	Multiple	MISD	MIMD

Fig. 2.4 – Classification de Flynn

Classe des modèles SISD

La classe des modèles SISD (Single Instruction stream - Single Data stream) est celle des machines mono-processeur classiques dans lesquelles un flux unique d'instructions est appliqué à un flux unique de données. La figure 2.5 présente le principe général de cette classe.

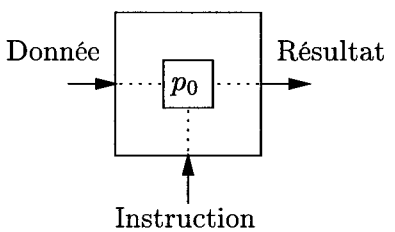

Fig. 2.5 – Classe des modèles SISD

Classe des modèles SIMD

La classe des modèles SIMD (Single Instruction stream - Multiple Data stream) est celle des machines parallèles équipées d'une unité de contrôle centralisée. Leur fonctionnement est de type synchrone. L'unité de contrôle envoie la même instruction à tous les processeurs de la machine. Ces derniers l'exécutent simultanément sur leur propre donnée et génèrent leur propre résultat. Le flux d'instructions est donc simple et le flux de données multiple. La figure 2.6 présente le principe général de cette classe.

Les processeurs de ces machines sont souvent peu puissants mais nombreux. Ce grand nombre de processeurs pose des problèmes au niveau de l'horloge interne de la machine. Le fonctionnement synchrone impose que tous les processeurs reçoivement simultanément le même top d'horloge. Ces difficultés techniques ont peu à peu conduit à l'abandon de ce modèle.

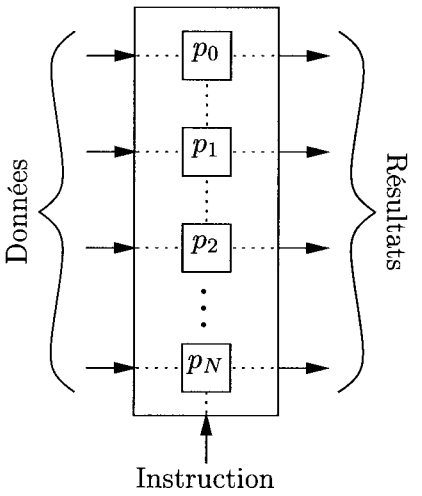

Fig. 2.6 – Classe des modèles SIMD

Classe des modèles MIMD

La classe des modèles MIMD (Multiple Instruction stream - Multiple Data stream) est celle des machines parallèles équipées de plusieurs unités de contrôle totalement indépendantes les unes des autres. Leur fonctionnement est de type asynchrone. Chaque processeur est autonome et gère son propre flux d'instructions et son propre flux de données. Les programmes qui s'exécutent sur ces processeurs peuvent être totalement différents. Le flux d'instructions et le flux de données sont donc multiples. La figure 2.7 présente le principe général de cette classe.

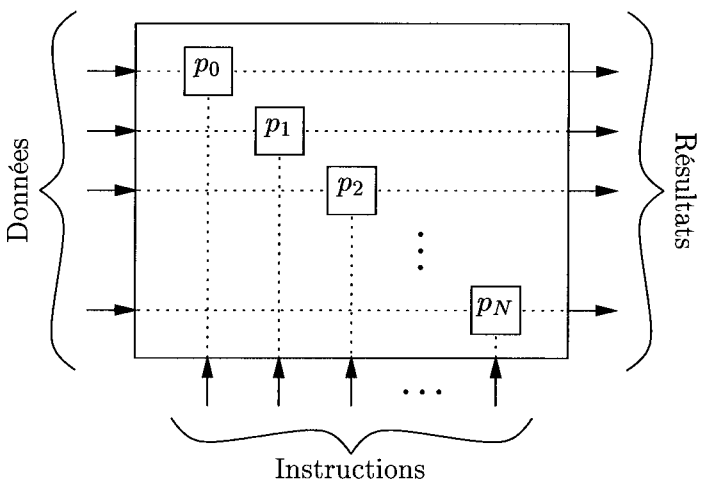

Fig. 2.7 – Classe des modèles MIMD

A l'inverse des machines SIMD, les processeurs des machines MIMD sont souvent puissants et peu nombreux. Le mode de fonctionnement asynchrone permet à chaque processeur de disposer de sa propre horloge interne. Ainsi dé-

barrassées de ce problème d'horloge, les machines massivement parallèles MIMD ont remplacé les machines massivement parallèles SIMD. Les machines MIMD sont à l'heure actuelle les machines parallèles les plus couramment rencontrées.

Le modèle SPMD (Single Program - Multiple Data) se rapproche à la fois des classes SIMD et MIMD. Dans la mesure où un même programme est appliqué sur des données différentes, il s'agit d'une généralisation de la classe SIMD. Dans la mesure où les processeurs sont libres d'exécuter des instructions différentes, le modèle SPMD se rapproche de la classe MIMD. Comme il n'existe pas de machine spécifique pour ce modèle, les machines MIMD font généralement double emploi. Le fonctionnement général de ce modèle est identique à celui de la figure 2.7, à l'exception près que les instructions sont ici remplacées par un même programme.

Classe des modèles MISD

Dans la réalité, la classe des modèles MISD (Multiple Instruction stream - Single Data stream) ne correspond à aucune machine. Par certaines similitudes, le modèle d'exécution pipeline, présenté ci dessous, se rapproche de cette classe. Cependant, la plupart des machines actuelles, parallèles ou non, comportent plusieurs pipelines sans pour autant faire partie de cette classe. La figure 2.8 présente le principe général de cette dernière.

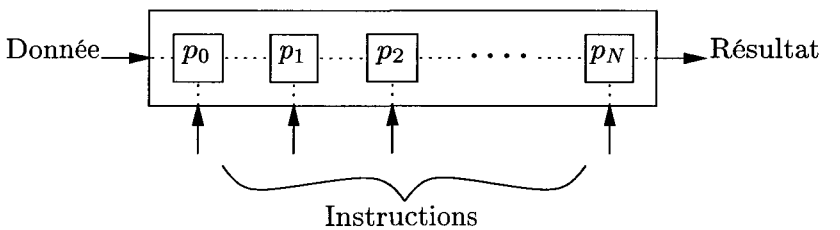

Fig. 2.8 – Classe des modèles MISD

Le modèle pipeline est bâti sur le principe de segmentation des instructions. Ce principe consiste à diviser l'instruction en étapes de durées égales. Ces étapes sont exécutées successivement. Chaque étage du pipeline est constitué d'une unité de contrôle et d'un processeur. Ce dernier exécute l'étape sur la donnée provenant de l'étage précédent et envoie le résultat vers l'étage suivant. Le but recherché est de délivrer un résultat à chaque top d'horloge. Ainsi, si le pipeline dispose de N étages et si les données sont en nombre suffisant, il est possible de gagner un facteur N par rapport à un processeur classique. Dans la réalité, il faut composer avec les ruptures de séquences qui obligent à vider tous les étages d'un pipeline avant de le réapprovisionner. Pour éviter cette chute brutale de performance, les processeurs actuels intégrent des mécanismes de prédiction de branchement qui permettent de prévoir, dans la mesure du possible, quelles seront les branches empruntées.

Les machines vectorielles sont une application du modèle pipeline. Ce dernier, de par son fonctionnement, se rapproche des classes SIMD et MIMD. Le

fonctionnement général de ce modèle est identique à celui de la figure 2.8, à l'exception près que les instructions sont ici remplacées par les étapes d'une même instruction.

2.3 Programmation parallèle

La programmation d'une machine parallèle est régie par un modéle de programmation qui est étroitement lié à son modèle d'exécution. Il y a encore quelques années, le programmeur pouvait choisir entre deux modèles : le modèle à parallélisme de tâches et le modèle à parallélisme de données. Aujourd'hui, l'apparition des machines parallèles à base de SMP semble ouvrir la voie pour un troisième modèle de programmation qui reste malgré tout difficile à définir car mélangeant deux types de parallélismes au sein d'une même application. Par la suite, nous tenterons de définir plus exactement ce qu'est ce nouveau modèle.

Quelque soit le modèle choisi, les deux conditions d'efficacité d'une application parallèle sont une bonne localité des données et un bon équilibrage de charge. La localité des données est une condition très importante quand l'application s'exécute sur une machine à mémoire répartie. Elle permet de limiter le volume des communications inter-processeurs, communications toujours très coûteuses sur ce type de machine. Son principe est de répartir les données de telle manière que chaque processeur dispose d'un maximum de données à traiter dans son espace mémoire local. L'équilibrage de charge permet de limiter les périodes d'inactivité des processeurs. Son principe est d'essayer d'attribuer au mieux les charges de calcul en fonction des caractéristiques de chaque processeur. Dans une machine parallèle homogène où tous les processeurs ont les mêmes caractéristiques, l'équilibrage de charge tend à attribuer la même charge de calcul à tous les processeurs.

La qualité d'une application parallèle est évaluée à partir de trois facteurs appelés facteur d'accélération, facteur d'efficacité et facteur de scalabilité. La loi d'Amdahl permet de borner ces trois facteurs.

Dans cette section, nous commençons par présenter les modèles classiques de programmation que sont les modèles à parallélisme de tâches et à parallélisme de données. Nous essayons ensuite de présenter les principales caractéristiques d'un troisième modèle de programmation que nous pouvons appeler modèle à parallélisme de tâches multi-threadées, en insistant bien sur le fait qu'à l'heure actuelle, un tel modèle ne possède pas encore de légitimité et que les recherches sur cette nouvelle voie n'en sont encore qu'à leurs débuts. Nous présentons ensuite les facteurs permettant de juger de la qualité d'une application parallèle ainsi que la loi d'Amdahl qui permet de les borner.

Modèle à parallélisme de tâches

Dans ce modèle, l'application est découpée en une suite de tâches élémentaires à exécuter. Ces tâches sont ensuite affectées aux différents processeurs disponibles. Les contraintes de ce modèle sont liées aux dépendances entre tâches. Elles imposent l'utilisation de barrières de synchronisation qui permettent d'attendre la fin de certaines tâches avant l'exécution de nouvelles tâches.

Ce modèle est adapté aux machines MIMD. Il ne nécessite pas de langages spécialisés et combine des langages standard tels que Fortran ou C, avec des bibliothèques de communication. Ces bibliothèques contiennent des primitives nécessaires à la coopération de tâches réparties sur un ensemble de processeurs. Les bibliothèques de communication standard les plus connues sont PVM (Parallel Virtual Machine) [35] et MPI (Message Passing Interface) [36].

Le modèle à parallélisme de tâches laisse le soin au programmeur de placer ses tâches et de synchroniser leur exécution. Il permet donc d'obtenir des bonnes performances mais sa maîtrise reste complexe.

Modèle à parallélisme de données

Le principe de ce modèle est d'appliquer un même traitement à un groupe de données homogènes, encore appelé tableau. Ce tableau est divisé en sous-tableaux qui sont répartis sur l'ensemble des processeurs. Durant l'exécution de l'application, une même instruction est appliquée à chaque élément des sous-tableaux. Le flux d'instructions est donc unique, ce qui permet d'éviter les problèmes de synchronisation du modèle à parallélisme de tâches. De fait, la structure de l'application reste quasi inchangée par rapport à la version s'exécutant sur une machine mono-processeur, le parallélisme n'apparaissant qu'au niveau des opérations menées sur les tableaux.

Ce modèle de programmation est adapté à tous les types d'architectures puisqu'un traitement de tableaux peut s'effectuer aussi bien en pipeline qu'en SIMD ou en SPMD.

Le parallélisme de données est un concept de plus haut niveau que le parallélisme de tâches. Son implémentation s'effectue par l'intermédiaire de langages spécialisés. Ces langages sont des extensions parallèles de langages standard tels que Fortran ou C. propriétaires à l'origine, ils ont fait l'objet de normalisations afin d'améliorer la portabilité des applications sur des plateformes différentes. L'exemple le plus connu est HPF (High Performance Fortran) [38]. Grâce à cette portabilité, le programmeur n'a plus à se soucier de la machine cible, l'adaptation étant réalisée par le compilateur. Cette portabilité et le fait que ces langages spécialisés soient des extensions de langages fortement connus de la communauté scientifique, simplifient l'écriture et la maintenance des applications parallèles. La contrepartie de cette simplicité est que les performances de l'application résultante sont liées à la qualité du compilateur utilisé et à la finesse des optimisations mises en oeuvre. Ces optimisations sont d'autant meilleures que le compilateur dispose d'un maximum d'informations au moment de la compilation. Ceci explique le fait que pour certaines applications spécifiques dont les informations ne sont disponibles qu'en cours d'exécution, le modèle par parallélisme de tâche soit préféré.

Modèle à parallélisme de tâches multi-threadées

Le thread (ou procesus léger) est l'entité de base pour la programmation des machines SMP. Pour expliquer ce qu'est le thread, nous allons commencer par rappeler brièvement ce qu'est le processus. Considérons pour cela une machine

mono-processeur classique et munie d'un système d'exploitation multitâches tel qu'Unix. Dans un tel système, les tâches sont encore appelées processus. Un processus est caractérisé, entre-autres, par des segments mémoires et par l'état des registres internes de la machine. Ces informations forment le contexte du processus. Lorsque ce dernier s'exécute, il est maître de toutes les ressources de la machine et donc du processeur. Supposons qu'à un instant donné, ce processus soit bloqué, par exemple dans l'attente d'une opération d'entrées-sorties. Le blocage est alors détecté par le système, qui, pour exploiter au mieux les ressources de la machine, va attribuer le processeur à un autre processus prêt à s'exécuter ou à reprendre le cours de son exécution. Cette opération suppose des sauvegardes et des restaurations de contextes. Il s'agit d'une opération relativement lourde qui dégrade d'autant les performances du processus bloqué. Supposons que dans le code de ce dernier, d'autres actions soient exécutables pendant l'attente provoquée par l'opération d'entrées-sorties. Le code du processus étant séquentiel, ces actions ne peuvent être déclenchées. Le thread permet de remédier à cette situation. Il s'agit d'un sous ensemble du code du processus qui peut être activé dès la détection du blocage d'un autre thread. Dès lors, même en cas de blocage d'un thread, le processus reste maître du processeur. Il évite ainsi les opérations de changement de contexte, ce qui améliore d'autant ses performances.

Sur les machines SMP, les threads peuvent être exécutés en parallèle et nous parlons alors de pthreads. Leur manipulation s'effectue par l'intermédiaire de bibliothèques proposant des mécanismes de synchronisation et d'exécution à distance. Cependant, ces dernières réclame une dextérité de programmation qui est loin d'être négligeable. OpenMP [51] est un langage très récent qui a été spécialement conçu pour simplifier la programmation multi-threadée sur ce type de machines. Comme HPF pour les données d'une application, OpenMP propose des mécanismes de haut niveau permettant de répartir les calculs de cette application.

Les choses se compliquent pour les machines à base de SMP. Quand elles ne sont pas CC-NUMA, c'est notamment le cas de la SP3 d'IBM, la seule façon de les programmer est de lancer plusieurs tâches multi-threadées sur les noeuds de calcul et de les faire communiquer entre elles par l'intermédiaire de messages. Dès lors, dans le cas général, nous pouvons définir le modèle de programmation de ces machines comme un modèle à parallélisme de tâches multi-threadées. Des travaux actuellement en cours portent sur l'hybridation MPI-OpenMP [30]. A ce jour (à l'exception de versions propriétaires telles que celle de Silicon Graphics), OpenMP ne dispose toujours pas de mécanismes lui permettant de répartir les données sur un noeud de calcul. Or, de tels mécanismes sont indispensables pour certains types d'applications. Il s'en suit une mauvaise utilisation des mémoires cache qui dégrade d'autant les performances de l'application. Pour contourner le problèmes, ces applications se tournent parfois vers un couplage MPI-HPF avec des compilateurs HPF capables de générer du code multi-threadé. Bien que les recherches dans ce domaine en soient également à leurs débuts, des compilateurs HPF tels qu'ADAPTOR [1] commencent à générer de pareils codes.

2.3.1 Evaluation des performances

La qualité d'une application séquentielle est généralement évaluée en fonction de son temps d'exécution. La qualité d'une application parallèle est évaluée de manière plus complexe en fonction de trois facteurs appelés facteur d'accélération, facteur d'efficacité et facteur de scalabilité.

Facteur d'accélération

Pour un problème de taille N, le facteur d'accélération est le rapport des temps utilisés par le meilleur algorithme séquentiel et le meilleur algorithme parallèle pour résoudre ce problème. Si $t_1(N)$ désigne le meilleur temps séquentiel et si $t_P(N)$ désigne le meilleur temps parallèle sur P processeurs, le facteur d'accélération est donné par :

$$s(N, P) = \frac{t_1(N)}{t_P(N)} \tag{2.3.1}$$

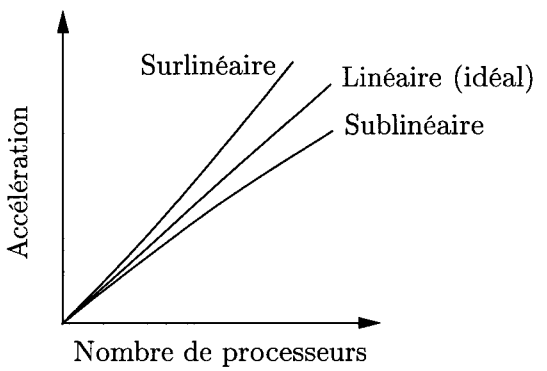

Fig. 2.9 – Facteur d'accélération

La figure 2.9 présente les trois cas possibles pour le facteur d'accélération. Celui ci peut être :

- linéaire avec $s(N, P) = P$. Il s'agit d'un cas idéal dans la mesure où les processeurs ne font que du calcul et pas de communications;
- sublinéaire avec $s(N, P) \leq P$. Il s'agit du cas le plus fréquemment rencontré dans la mesure ou les processeurs calculent et communiquent entre eux;
- surlinéaire avec $s(N, P) \geq P$. Il s'agit d'un cas plus rare puisque l'application présente des comportements très différents selon le nombre de processeurs.

Facteur d'efficacité

Le facteur d'efficacité permet d'établir le taux d'utilisation des processeurs. Il normalise le facteur d'accélération. En reprenant les notations précédentes, ce

facteur est donné par :

$$e(N,P) = \frac{s(N,P)}{P} = \frac{t_1(N)}{P\,t_P(N)} \tag{2.3.2}$$

De manière analogue à la figure 2.9, la figure 2.10 présente les trois cas possibles pour l'efficacité.

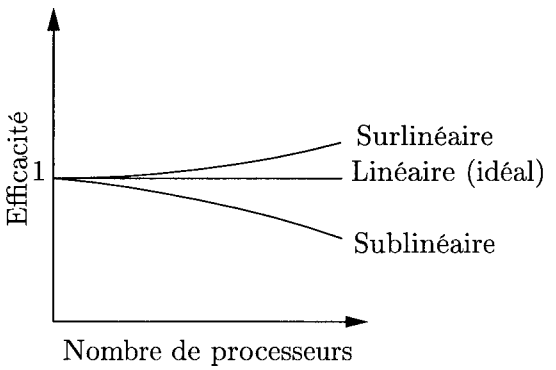

Fig. 2.10 – Facteur d'efficacité

La qualité d'une application parallèle est d'autant meilleure que son efficacité est proche de 1, c'est à dire que son accélération est proche du cas linéaire.

Facteur de scalabilité

Le facteur de scalabilité représente la capacité de l'application à traiter des problèmes de tailles de plus en plus importantes avec un nombre de processeurs en rapport avec ces tailles. Il s'agit en fait du facteur d'efficacité mesuré en faisant varier proportionnellement la taille du problème et le nombre de processeurs. Plus l'efficacité est proche du cas linéaire et plus l'application possède une bonne scalabilité.

Loi d'Amdhal

Toute application parallèle contient une zone de code séquentielle qui ne peut être parallélisée. Pour un problème de taille N, notons $t_{\text{séq}}(N)$ le temps d'exécution de la zone de code séquentielle et $t_{\text{par}}(N)$ le temps d'exécution de la zone de code parallèle. Le temps d'exécution de cette application peut alors s'écrire sous la forme :

$$t_1(N) = t_{\text{séq}}(N) + t_{\text{par}}(N) \tag{2.3.3}$$

sur une machine mono-processeur, et sous la forme :

$$t_P(N) = t_{\text{séq}}(N) + \frac{t_{\text{par}}(N)}{P} \tag{2.3.4}$$

sur une machine parallèle à P processeurs.

Si $f(N)$ désigne la proportion de la zone de code non-parallélisable dans l'application complète, nous pouvons également écrire :

$$t_{\text{séq}}(N) = f(N)\, t_1(N) \tag{2.3.5}$$
$$t_{\text{par}}(N) = (1 - f(N))\, t_1(N) \tag{2.3.6}$$

Le facteur d'accélération de cette application est égal à :

$$s(N, P) = \frac{t_{\text{séq}}(N) + t_{\text{par}}(N)}{t_{\text{séq}}(N) + \frac{t_{\text{par}}(N)}{P}} \leq \frac{t_1(N)}{t_{\text{séq}}(N)} \tag{2.3.7}$$

Nous pouvons alors écrire :

$$s(N, P) = \frac{1}{f(N) + \frac{1 - f(N)}{P}} \leq \frac{1}{f(N)} \tag{2.3.8}$$
$$e(N, P) = \frac{1}{1 + (P - 1)\, f(N)} \tag{2.3.9}$$

Cela montre que si la zone de code non-parallélisable représente par exemple 15% de l'application complète, le facteur d'accélération ne peut excéder 6,67, quelque soit le nombre de processeurs employés. Notons que de par sa définition, la loi d'Amdhal exclut les accélérations surlinéaires. Ces dernières peuvent résulter d'une meilleure utilisation des mémoires cache. Elles peuvent également résulter des algorithmes eux-mêmes. Citons par exemple le cas d'une méthode itérative de résolution de systèmes d'équations avec un préconditionnement par blocs, chaque bloc correspondant à un processeur.

2.4 Conclusion

Pour répondre aux besoins en puissance de calcul de la communauté scientifique, le parallélisme s'est considérablement développé depuis 1985 [29], au point d'être omniprésent aujourd'hui sous une forme ou sous une autre. Ces dernières années, les machines à mémoire partagée et les machines à mémoire répartie ont dominé le marché. À l'heure actuelle, nous assistons à l'émergence d'un nouveau type de machines qui tente de cumuler les avantages des mémoires partagées et réparties : les machines à base de SMP.

Que ces machines soient à mémoire partagée, à mémoire répartie, SMP ou à base de SMP, leur maîtrise passe par une bonne connaissance de leur architecture et de leur modèle d'exécution. Ce modèle permet de décrire les relations entre les instructions et les données auxquelles elles s'appliquent. Les utilisateurs du parallélisme sont généralement issus de milieux scientifiques autres que celui de l'informatique et cette connaissance leur paraît difficile à acquérir. Des classifications telles que celle de Flynn [34], leur permettent de mieux appréhender les mécanismes élémentaires de ces machines. Les différents types de machines parallèles ainsi que les différents modèles d'exécution ont été présentés dans la section 2.2.

Il n'existe pas de modèle de programmation universel pour les machines parallèles. Généralement, les applications parallèles combinent les différents modèles pour résoudre un problème donné. Dans le modèle à parallélisme de tâches,

le programmeur est responsable de la distribution et de la synchronisation de ses tâches. Ce modèle permet d'obtenir de bonnes performances mais sa maîtrise est complexe. Le modèle à parallélisme de données utilise un concept de plus haut niveau que celui à parallélisme de tâches. Dans ce cas, le langage et le compilateur déchargent le programmeur d'une partie du travail qu'il devrait accomplir dans le modèle à parallélisme de tâches. Il permet donc de simplifier l'écriture et la maintenance des codes parallèles, ce qui représente un atout non-négligeable pour la communauté scientifique non-spécialiste. La contrepartie est que les performances de l'application résultante deviennent dépendantes du compilateur utilisé. Le modèle à parallélisme de tâches multi-threadées s'adresse aux machines à base de SMP et constitue une nouvelle voie de recherche. Il permet d'accroître l'efficacité du modèle à parallélisme de tâches en implantant un second niveau de parallélisme au sein de la tâche elle-même. La contrepartie de cette efficacité est un accroissement important de la complexité de programmation que des outils tels qu'OpenMP [51] tentent de réduire.

La portabilité des codes parallèles est un souci permanent. De par la normalisation des langages parallèles et des bibliothèques de passage de messages, les modèles de programmation à parallélisme de tâches et de données permettaient plus ou moins de garantir cette portabilité sur les machines à mémoire partagée ou répartie, le choix de l'un ou de l'autre étant souvent dicté par des considérations de facilité de développement et de capacité d'évolution des applications concernées. Le problème qui se pose aujourd'hui est celui des machines à base de SMP qui ne sont pas CC-NUMA. Comme nous l'avons dit précédemment, le modèle de programmation de ces machines peut être qualifié de modèle à parallélisme de tâches multi-threadées. Les applications construites sur ce modèle peuvent tenter d'utiliser des couplages MPI-OpenMP [30] ou MPI-HPF avec des compilateurs HPF qui soient capables de générer du code multi-threadé [52, 19]. Le concept d'OpenMP est un concept de moins haut niveau que celui de HPF car il se limite au seul contrôle du code de l'application. Or, pour certains types d'applications, un tel contrôle n'est pas suffisant et doit être accompagné d'un contrôle de la répartition des données, sous peine d'une mauvaise utilisation des mémoires cache. Contrairement aux compilateurs OpenMP, les compilateurs HPF sont totalement libres dans le choix du modèle d'exécution. Ce modèle peut donc être SPMD, SPMD multi-threadé ou autres sans que le programmeur ait quelque chose à modifier dans son application. Dès lors, sous réserve de disposer de compilateurs HPF générant du code multi-threadé relativement efficace, nous devinons l'intérêt d'un langage tel que HPF dans le cas des machines à base de SMP. Les recherches dans cette voie ont d'ailleurs commencé et certains compilateurs HPF tels qu'ADAPTOR [1] génèrent déjà du code multi-threadé.

Quels que soient les modèles de programmation utilisés, la localité des données et l'équilibrage de charge sont les deux conditions de performance d'une application parallèle. Cette performance est évaluée en fonction de trois critères appelés facteur d'accélération, facteur d'efficacité et facteur de scalabilité. Ces facteurs sont bornés par la loi d'Amdhal. Les différents modèles de programmation ainsi que les facteurs d'évaluation et la loi d'Amdhal ont été présentés dans la section 2.3.

En simulation numérique, quels que soient les domaines physiques concernés

et quelles que soient les méthodes de discrétisation employées, il nous faut toujours résoudre des systèmes d'équations linéaires. Dans le cas des simulations réalistes intégrant un grand nombre de paramètres, ces systèmes sont généralement de très grande taille et ne peuvent être résolus que sur des calculateurs parallèles. Lorsque les matrices de ces systèmes possèdent des structures par blocs, il est possible d'attribuer ces derniers aux différents processeurs de la machine et ainsi réduire au minimum les volumes de communications échangés lors de la résolution. Malheureusement, outre le fait d'être creuses et souvent à diagonale dominante, les matrices de ces systèmes ne présentent généralement pas une telle structure. Dans ces conditions, les résultats obtenus par les méthodes de résolution classiques, qu'elles soient directes ou itératives, sont souvent décevants en raison des volumes de communications générés trop importants. Pour remédier à ce problème, l'analyse numérique à developpé toute une famille de méthodes permettant de construire des matrices par blocs, structure parfaitement adaptée aux architectures parallèles, ainsi que des préconditionneurs pour ces matrices. Ces méthodes, dites méthodes additives de Schwarz, sont construites à partir d'un mélange entre l'algèbre linéaire et certaines propriétés mathématiques des opérateurs différentiels concernés. Ces méthodes ne sont qu'un sous-ensemble d'un ensemble beaucoup plus vaste, celui des méthodes dites de Schwarz. Ces dernières font l'objet du chapitre suivant.

Chapitre 3

Méthodes de Schwarz

3.1 Introduction

Le corollaire de la simulation numérique est un système d'équations linéaires à résoudre. Dans le cas d'une simulation réaliste, la taille de ce système est suffisamment importante pour que sa résolution ne puisse se faire que sur des calculateurs parallèles. La matrice de ce système est creuse, parfois symétrique, souvent à diagonale dominante, mais ne présente généralement pas une structure particulière qui permettrait d'exploiter efficacement les architectures parallèles. Cette inadaptation fait que les résultats obtenus par les méthodes de résolution classiques, directes ou itératives, sont souvent décevants en raison des volumes de communications générés trop importants. Les méthodes additives de Schwarz ont été développées par l'analyse numérique pour résoudre ce problème. Elles constituent un mélange entre l'algèbre linéaire et certaines propriétés mathématiques des opérateurs différentiels concernés. Les matrices par blocs et les préconditionneurs associés que ces méthodes permettent de construire, sont parfaitement adaptés aux machines parallèles. Les méthodes additives de Schwarz font partie d'un ensemble beaucoup plus vaste comprenant également les méthodes dites multiplicatives de Schwarz. Bien que le fonctionnement intrinsèque de ces dernières soit de nature séquentielle, de nombreux préconditionneurs construits à partir des méthodes additives intègrent un aspect issu des méthodes multiplicatives.

Les méthodes de Schwarz sont très souvent employées dans les codes parallèles de simulation pour des domaines aussi variés que l'élasticité linéaire [61], la mécanique des fluides [21] ou les semi-conducteurs [26]. Leur principe est de décomposer l'espace d'approximation en sous-espaces, de calculer puis de corriger une solution globale approchée sur chacun de ces sous-espaces [64]. Le principe de décomposition et de correction [65] permet de classer les méthodes de Schwarz en deux catégories bien distinctes. La catégorie des méthodes additives regroupe les méthodes dans lesquelles la solution globale approchée est corrigée simultanément (en parallèle) sur l'ensemble des sous-espaces. La nature de ces méthodes est analogue à celle de la méthode de Jacobi. La catégorie des méthodes multiplicatives regroupe les méthodes dans lesquelles cette solution globale approchée n'est corrigée que sur un seul sous-espace à la fois. La nature

de ces méthodes est analogue à celle de la méthode de Gauss-Siedel.

Les bases hiérarchiques [46], les méthodes multi-grilles [40] et les préconditionneurs multi-niveaux [39] sont des méthodes multiplicatives de Schwarz encore appelées méthodes de Schwarz multi-niveaux. Dans ce type de méthodes, le maillage du domaine d'étude est raffiné pour donner naissance à une hiérarchie de maillages imbriqués. A chaque maillage est associé un espace d'éléments finis. La décomposition de l'espace d'approximation est basée sur cette suite d'espaces d'éléments finis. Les méthodes de sous-domaines [60] sont des méthodes additives de Schwarz. La décomposition de l'espace d'approximation est ici basée sur la décomposition du domaine d'étude en sous-domaines plus petits. Lorsque les sous-domaines se recouvrent, nous parlons de méthodes de sous-domaines avec recouvrement. Dans ce cas, la restriction de la solution globale approchée dans un sous-domaine est utilisée comme condition à la limite homogène de Dirichlet pour le calcul des solutions dans les sous-domaines voisins. Lorsque les sous-domaines ne se recouvrent pas, des conditions de continuité sont imposées sur les interfaces entre sous-domaines. Des conditions homogènes de Dirichlet conduisent à la méthode du complément de Schur [26], tandis que des conditions de Neumann conduisent à la méthode FETI [15]. Des conditions aux limites plus complexes [50] peuvent également être utilisées.

Les méthodes de Schwarz sont initialement conçues pour traiter les problèmes elliptiques du second ordre. Les propriétés mathématiques des opérateurs différentiels mis en jeu font qu'il existe une base constituée des vecteurs propres de l'opérateur dans laquelle sa forme matricielle est diagonale. La décomposition de l'espace d'approximation est alors implicite. Il n'est bien sûr pas possible de calculer ces vecteurs propres car une telle opération serait aussi coûteuse que la résolution elle-même. Les méthodes de Schwarz utilisent des approximations de ces vecteurs propres et plus exactement des approximations des opérateurs de projection dans les sous-espaces engendrés par ces vecteurs. Le cas des problèmes paraboliques et hyperboliques est plus complexe. Dans le domaine électromagnétique, la décomposition de Helmholtz permet maintenant de traiter les formulations magnétodynamiques du chapitre 1 par des méthodes additives et multiplicatives de Schwarz. Ces travaux sont relativement récents [39, 40, 41, 64], ce qui explique en partie le retard pris par le génie électrique en matière de calcul numérique parallèle [42]. Les problèmes hyperboliques peuvent, quand à eux, être traités par des méthodes de sous-domaines sans recouvrement, moyennant l'utilisation de conditions de transmission particulières [14] sur les interfaces en sous-domaines.

Dans ce chapitre, nous commençons par présenter un cadre abstrait permettant d'établir la convergence des méthodes de Schwarz dans le cas des problèmes elliptiques du second ordre. Nous en profitons pour présenter la décomposition de Helmohltz qui permet de ramener les formulations magnétodynamiques du chapitre 1, à deux problèmes elliptiques du second ordre. Nous présentons ensuite les principes généraux et les limitations des méthodes multi-niveaux et des méthodes de sous-domaines.

3.2 Cadre abstrait

Dans cette section, nous commençons par présenter certaines caractéristiques mathématiques des opérateurs différentiels elliptiques du second ordre. Nous présentons ensuite le principe de décomposition et de correction puis un cadre abstrait permettant d'établir la convergence des méthodes de Schwarz dans le cas des problèmes ellipiques du second ordre [65, 66]. Nous présentons également la décomposition de Helmohltz qui permet d'appliquer ce cadre aux formulations magnétodynamiques du chapitre 1. Les notations employées pour les espaces fonctionnels sont celles de ce chapitre.

3.2.1 Problèmes elliptiques du second ordre

Considérons le problème de laplacien avec conditions aux limites homogènes de Dirichlet suivant :

$$\begin{cases} -\triangle u(x) &= f(x), \quad \forall\, x \in \Omega \\ u(x) &= 0, \quad \forall\, x \in \Gamma \end{cases} \tag{3.2.1}$$

Il s'agit d'un problème elliptique du second ordre que nous avons rencontré lors de la discrétisation des formulations magnétostatiques du chapitre 1 et dont la forme variationnelle est donnée par :

$$a(u, v) = (grad\, u, grad\, v) = (f, v), \quad \forall\, v \in E_h^0,\, u \in E_h^0 \tag{3.2.2}$$

Cette forme est symétrique, positive, continue et coercive (ou fortement elliptique), cette dernière propriété résultant du lemme de Poincaré [59] :

Lemme de Poincaré : Soit Ω un ouvert borné régulier. Il existe un réel positif β tel que :

$$\| v \|_{L^2} \leq \beta \| grad\, v \|_{L^2}\ , \quad \forall\, v \in E_h^0 \tag{3.2.3}$$

□

Dès lors, d'après le théorème de Lax-Milgram, la solution du problème variationnel existe et est unique [59] :

Théorème de Lax-Milgram : Soit V un espace Hilbertien, $a : V \times V \to R$ une forme bilinéaire continue, symétrique et coercive et $L : V \to R$ une forme linéaire continue. Il existe une unique solution u du problème variationnel :

$$a(u, v) = (L, v), \quad \forall\, v \in V,\, u \in V \tag{3.2.4}$$

□

Les propriétés de cette forme bilinéaire nous permettent d'appliquer le théorème suivant :

Théorème : Il existe une suite croissante de réels positifs λ_n, $n \in \mathbb{N}^*$, et une suite de fonctions $e_n \in E_h^0$, $n \in \mathbb{N}^*$, telle que :

$$-\triangle e_n = \lambda_n e_n \tag{3.2.5}$$

La suite (λ_n) tend vers $+\infty$ avec n, et la famille $(e_n)_{n\geq 1}$ forme une base hilbertienne de l'espace L^2.
□

La suite (λ_n), $n \in \mathbb{N}^*$, est la suite des valeurs propres de l'opérateur. De manière informelle, un opérateur différentiel du second ordre est dit elliptique s'il se comporte comme un laplacien, c'est à dire que la forme bilinéaire associée au problème variationnel qui le met en jeu est coercive.

Nous pouvons alors approximer la solution du problème 3.2.1 par :

$$u^N = \sum_{i=1}^{N} \frac{f_n}{\lambda_n} e_n \tag{3.2.6}$$

et prouver le résultat :

$$\| u - u^N \|_{L^2} \leq \frac{\| f \|_{L^2}}{\lambda_{N+1}} \tag{3.2.7}$$

Cela signifie que si la fonction f se décompose sur quelques vecteurs propres du laplacien, alors la solution u du problème 3.2.1 s'exprime elle aussi en fonction de ces vecteurs propres et ne dépend que très peu des degrés de liberté.

3.2.2 Méthodes additives et multiplicatives

Soit V un espace de Hilbert de dimension finie. Nous considérons le problème variationnel :

$$a(u, v) = (f, v), \quad \forall\, v \in V, u \in V \tag{3.2.8}$$

où $a : V \times V \to \mathbb{R}$ est une forme bilinéaire auto-adjointe définie positive, dont l'opérateur associé A est un opérateur différentiel elliptique du second ordre.

Une décomposition de V est une suite de sous-espaces $V_i \subset V$, telle que :

$$V = \sum_{i=0}^{J} V_i \tag{3.2.9}$$

où le symbole $\sum$ ne désigne pas forcément une somme directe.

Soit E_i l'injection canonique de V_i dans V. Nous définissons la restriction de A au sous-espace V_i par :

$$A_i = E_i^T \, A \, E_i \tag{3.2.10}$$

Soit B_i une approximation de A_i sur V_i et $T_i : V \to V_i$, l'opérateur de projection associé à V_i et défini par :

$$T_i = E_i \, B_i^{-1} \, E_i^T \, A \tag{3.2.11}$$

Nous remarquons qu'en choisissant $B_i = A_i$ sur chaque sous-espace V_i, l'opérateur T_i est l'opérateur de projection a-orthogonale sur V_i.

Nous définissons l'opérateur de propagation d'erreur sur V_i par :

$$E_i = (I - T_i) \tag{3.2.12}$$

Dans une méthode itérative, cet opérateur rend l'erreur othogonale à V_i. La correction apportée à la solution sur le sous-espace V_i est alors la projection de l'erreur sur ce sous-espace. C'est pourquoi nous parlons de méthodes de correction par sous-espaces.

L'opérateur additif de Schwarz et l'opérateur multiplicatif symétrique de Schwarz, sont respectivement définis par :

$$T_{as} = \sum_{i=0}^{J} T_i \tag{3.2.13}$$

$$T_{ms} = I - (I - T_0) \ldots (I - T_J)(I - T_J) \ldots (I - T_0) \tag{3.2.14}$$

et peuvent être utilisés comme préconditionneurs. Nous résolvons alors la nouvelle équation $T_{as}\, u = g$ ou $T_{ms}\, u = g$ par une méthode de type gradient conjugué, sans préconditionnement supplémentaire, en utilisant le produit scalaire $a(\cdot,\cdot)$ et avec un second membre g approprié. Le préconditionnement par l'opérateur additif de Schwarz et par l'opérateur multiplicatif de Schwarz, nécessite de résoudre, respectivement en parallèle et en séquentiel, $J+1$ systèmes linéaires de plus petites tailles avec l'opérateur B_i.

3.2.3 Théorie de convergence

Deux hypothèses permettent d'établir la convergence des méthodes additives et multiplicatives de Schwarz [41]. La première est relative à la stabilité de la décomposition par rapport à la norme définie par $a(\cdot,\cdot)$. La seconde est relative à la quasi orthogonalité des sous-espaces intervenant dans la décomposition.

Hypothèse : (Stabilité de la décomposition). Il existe une constante $C_0 > 0$ telle que :

$$inf\{ \sum_i a(v_i, v_i);\ \sum_i v_i = v,\ v_i \in V_i \} \leq C_0^2\, a(v,v), \quad \forall\, v \in V \tag{3.2.15}$$

□

Hypothèse : (Quasi orthogonalité des sous-espaces). Il existe des constantes $0 \leq \epsilon_{ij} \leq 1$ telles que les inégalités étendues de Cauchy-Schwarz suivantes soient vérifiées :

$$| a(u_i, v_j) | \leq \epsilon_{ij} \sqrt{a(u_i, u_i)} \sqrt{a(v_j, v_j)}, \quad \forall\, (u_i, v_j) \in V_i \times V_j, 1 \leq i, j \leq J \tag{3.2.16}$$

□

Soit la matrice $E = (\epsilon_{ij}) \in \mathbb{R}^n$ et $\rho(E)$ son rayon spectral. Les deux hypothèses précédentes conduisent au lemme :

Lemme : Si les deux hypothèses précédentes sont vérifiées, alors nous avons :

$$C_0^{-2} a(u, u) \leq a(T_{as} u, u) \leq (\rho(E) + 1)\, a(u, u), \quad \forall\, u \in V \tag{3.2.17}$$

$$(1 + 2\rho^2(E))^{-1} C_0^{-2} a(u, u) \leq a(T_{ms} u, u) \leq a(u, u), \quad \forall\, u \in V \tag{3.2.18}$$

□

La première hypothèse permet de mesurer la stabilité de la décomposition, mais le paramètre C_0 est souvent difficile à estimer. La seconde hypothèse mesure les interactions entre les sous-espaces V_i à l'aide des angles entre ces sous-espaces pour le produit scalaire $a(\cdot, \cdot)$. Nous remarquons au passage que cette hypothèse ne tient pas compte du sous-espace V_0. Ce sous-espace, dit global, interagit avec tous les autres sous-espaces. Il correspond aux frontières entre les sous-domaines dans les méthodes de décomposition de domaines et à la grille de plus grand diamètre dans les méthodes multi-niveaux. Quant au lemme, son rôle est de majorer les rayons spectraux de $B_i^{-1} A_i$.

3.2.4 Décomposition de Helmholtz

Considérons la forme bilinéaire auto-adjointe définie positive suivante :

$$a(\mathbf{u}, \mathbf{v}) = (\mathbf{u}, \mathbf{v}) + \eta(\mathbf{rot}\ \mathbf{u}, \mathbf{rot}\ \mathbf{v}), \quad \forall\, (\mathbf{u}, \mathbf{v}) \in \mathbf{E^1} \times \mathbf{E^1} \tag{3.2.19}$$

où η est un paramètre réel strictement positif. Une telle forme apparaît lors de la discrétisation des formulations magnétodynamiques du chapitre 1. De cette forme, nous pouvons extraire un opérateur linéaire symétrique d'un espace fonctionnel dans son espace dual :

$$(\mathbf{L}\ \mathbf{u}, \mathbf{v}) = a(\mathbf{u}, \mathbf{v}), \quad \forall\, \mathbf{v} \in \mathbf{E^1}, \mathbf{u} \in \mathbf{E^1} \tag{3.2.20}$$

avec :

$$\mathbf{L} = \mathbf{I} + \eta\ \mathbf{rot}\ \mathbf{rot} \tag{3.2.21}$$

Il ne s'agit pas d'un opérateur elliptique du second ordre. Pour le montrer, considérons la relation vectorielle suivante :

$$\mathbf{rot}\ \mathbf{rot}\ \mathbf{u} = grad\ \mathbf{div}\ \mathbf{u} - \triangle \mathbf{u} \tag{3.2.22}$$

et la décomposition orthogonale de l'espace $\mathbf{L^2}$ mise en évidence par le diagramme de De Rham continu du chapitre 1. Si nous appliquons cet opérateur à un élément du sous-espace $grad\ E^0$, nous constatons qu'il se comporte comme l'opérateur identité $\mathbf{I}$. Si nous l'appliquons à un élément appartenant à l'orthogonal du sous-espace $grad\ E^0$, nous constatons qu'il se comporte comme l'opérateur :

$$\mathbf{L} = \mathbf{I} + \eta(grad\ \mathbf{div} - \triangle) \tag{3.2.23}$$

En se restreignant aux potentiels vecteurs à divergence nulle, nous pouvons dire que l'opérateur $\mathbf{L}$ se comporte comme l'opérateur elliptique du second ordre

$(\mathbf{I} - \eta \triangle)$ sur l'orthogonal du noyau de l'opérateur différentiel rotationnel. Si nous introduisons deux éléments de ce noyau dans la relation 3.2.19, nous obtenons :

$$(grad\, u, grad\, v) = (f, v), \quad \forall\, v \in E^0, u \in E^0 \tag{3.2.24}$$

Nous retrouvons la forme variationnelle du problème elliptique du second ordre 3.2.1. Dès lors, nous pouvons dire qu'en ne considérant que les potentiels vecteurs à divergence nulle, l'opérateur $\mathbf{L}$ se comporte comme un opérateur elliptique du second ordre sur le noyau de l'opérateur différentiel rotationnel et sur l'orthogonal de ce dernier. Cette décomposition entre le noyau de l'opérateur et son orthogonal est appelée décomposition de Helmholtz. Elle permet d'appliquer les méthodes de Schwarz aux formulations magnétodynamiques du chapitre 1.

3.3 Méthodes multi-niveaux et de sous-domaines

La différence entre les méthodes multi-niveaux et les méthodes de sous-domaines, provient de la décomposition de l'espace d'approximation. Dans les méthodes multi-niveaux, le maillage du domaine d'étude est raffiné pour donner naissance à une hiérarchie de maillages imbriqués les uns dans les autres. A chaque maillage est associé un espace d'éléments finis. La décomposition de l'espace d'approximation est basée sur cette suite d'espaces d'éléments finis [39, 62]. Dans les méthodes de sous-domaines, la décomposition de l'espace d'approximation est basée sur la décomposition du domaine d'étude en sous-domaines plus petits [60].

Dans cette section, nous présentons les principes généraux de ces méthodes ainsi que leurs limitations.

3.3.1 Méthodes multi-niveaux

Les bases hiérarchiques [46], les méthodes multi-grilles [40] et les préconditionneurs multi-niveaux [39] sont des méthodes de Schwarz multi-niveaux. Ces méthodes sont réservées aux problèmes elliptiques du second ordre. Partant d'un maillage initial du domaine d'étude, ces dernières amorcent un processus de raffinement récursif qui conduit à la création d'une hiérachie de maillages imbriqués les uns dans les autres. A chaque maillage est associé un espace d'éléments finis. Dès lors nous avons $V_0 \subset V_1 \subset ... \subset V_J = V$ et donc $V = \sum_{i=0}^{J} V_i$. La figure 3.1 illustre cette décomposition dans laquelle une grille grossière de diamètre H est raffinée en une grille plus fine de diamètre h, $h < H$.

Ces méthodes permettent de construire des préconditionneurs multiplicatifs à partir des algorithmes de type V-cycle des méthodes multi-grilles [46], ou additifs comme BPX [17]. Ces préconditionneurs choisissent systématiquement $B_0 = A_0$ afin d'approcher les valeurs propres les plus basses de l'opérateur différentiel. Ils utilisent des opérateurs d'interpolation et de restriction pour passer d'un niveau à un autre. De tels opérateurs ne peuvent être définis que sur des maillages quasi-uniformes. Moyennant cette quasi-uniformité, il est possible de

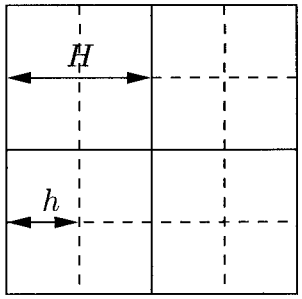

Fig. 3.1 – Hiérachie de maillages quasi-uniformes

montrer que le conditionnement de l'opérateur différentiel préconditionné ne dépend pas du niveau de raffinement J. Notons que dans ces méthodes, le nombre d'itérations du gradient conjugué est très sensible aux sauts de coefficients de l'opérateur différentiel.

Ces méthodes sont très efficaces en dimension deux mais ne se généralisent pas facilement en dimension trois. Les problèmes qui se posent sont liés à la construction de maillages imbriqués quasi-uniformes et à la difficulté de définir des opérateurs d'interpolation et de restriction.

3.3.2 Méthodes de sous-domaines

Les méthodes de sous-domaines sont d'autres méthodes de Schwarz dans lesquelles la décomposition du domaine d'approximation est basée sur une décomposition du domaine d'étude en sous-domaines plus petits. Les méthodes de sous-domaines se subdivisent en méthodes avec recouvrement et méthodes sans recouvrement. Les méthodes avec recouvrement sont réservées aux problèmes elliptiques du second ordre. Les méthodes sans recouvrement permettent de gérer d'autres types de problèmes moyennant des conditions particulières de transmission sur les interfaces entre sous-domaines [14].

Méthodes avec recouvrement

Dans les méthodes avec recouvrement, la restriction de la solution globale approchée dans un domaine est utilisée comme condition à la limite homogène de Dirichlet pour le calcul des solutions dans les sous-domaines voisins. La figure 3.2 présente la décomposition du domaine d'étude en deux sous-domaines qui se recouvrent.

La méthode alternée de Schwarz (1870) est à l'origine de ces méthodes. Dans ce schéma de type multiplicatif, la solution calculée à l'étape P dans un sous-domaine est utilisée comme condition à la limite homogène de Dirichlet pour le calcul des solutions à l'étape P dans les sous-domaines voisins.

Ces méthodes sont utilisées pour construire des préconditionneurs additifs. Dans ce schéma, la solution calculée à l'étape P dans un sous-domaine est utilisée comme condition à la limite homogène de Dirichlet pour le calcul des solutions à l'étape $P + 1$ dans les sous-domaines voisins. La solution affectée à l'étape

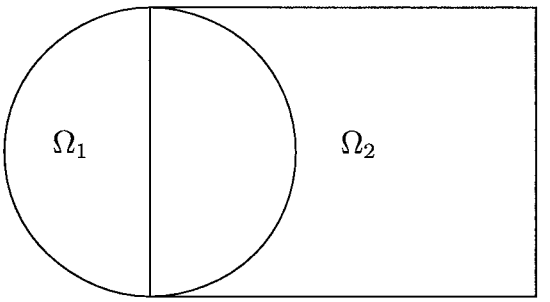

Fig. 3.2 – Décomposition avec recouvrement

$P+1$ à une zone de recouvrement est une combinaison des solutions obtenues à l'étape $P+1$ dans les sous-domaines voisins [46].

Moyennant des conditions sur la régularité des sous-domaines, sur le nombre moyen de voisins pour un sous-domaine et sur le nombre moyen de sous-domaines pour un point du domaine d'étude, il est possible de montrer que le conditionnement de l'opérateur différentiel préconditionné est d'autant meilleur que le recouvrement est important, mais qu'il se dégrade d'autant plus vite que le nombre de sous-domaines augmente [41]. Cette dégradation est dûe au fait que les communications entre les sous-domaines n'existent qu'au niveau local, entre voisins. Il est alors nécessaire d'introduire un problème dit global afin d'assurer un transfert d'information entre tous les sous-domaines. Ce problème global est associé au sous-espace V_0. Nous parlons alors de méthodes avec recouvrement à deux niveaux. Avec ces méthodes, le conditionnement de l'opérateur différentiel préconditionné ne dépend plus que du recouvrement entre les sous-domaines [64].

Méthodes sans recouvrement

Le principe des méthodes sans recouvrement est de résoudre la restriction du problème global sur chaque sous-domaine en imposant des conditions de continuité adéquates sur les interfaces entre sous-domaines, afin que chaque problème local soit bien posé. Ces méthodes sont utilisées pour construire des préconditionneurs additifs. La figure 3.3 présente la décomposition du domaine d'étude en deux sous-domaines qui ne se recouvrent pas.

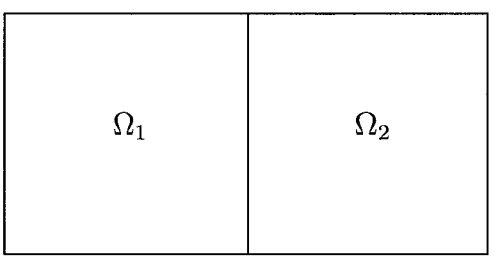

Fig. 3.3 – Décomposition sans recouvrement

Les conditions de continuité les plus couramment rencontrées sont des condi-

tions homogènes de Dirichlet et conduisent à la méthode du complément de Schur [26]. Dans cette méthode, l'élimination des inconnues internes aux sous-domaines permet de se ramener à un problème sur les interfaces entre sous-domaines. Tout comme pour les méthodes avec recouvrement, un problème global lié au sous-espace V_0 doit être introduit. Il est alors possible de montrer que le conditionnement de l'opérateur différentiel préconditionné est d'autant meilleur que les matrices B_i et A_i sont spectralement proches pour $0 \leq i \leq J$.

Il est possible de choisir $B_i = A_i$ pour $1 \leq i \leq J$. Cette solution requiert des méthodes de résolution exactes à l'intérieur des sous-domaines, ce qui n'est pas forcément souhaitable. Par contre, il est rarement possible de choisir $B_0 = A_0$ car A_0 est une matrice pleine qui, généralement, n'est pas stockée explicitement. De nombreux préconditionneurs tels que BPS [16] ont été étudiés pour B_0.

Tout comme pour les méthodes multi-niveaux, la méthode de Schur est très efficace en dimension deux mais ne se généralise pas facilement en dimension trois. Le problème posé est celui de l'opérateur d'interpolation sous-jacent utilisé par cette méthode pour passer de V à V_0. Une solution à ce problème consiste à construire l'espace V_0 à l'aide d'opérateurs de moyenne [62]. Notons que cette solution ne permet pas de préserver la somme directe dans la décomposition de l'espace d'approximation.

Il est possible d'employer d'autres conditions de continuité sur les interfaces. Ainsi, des conditions de Neumann conduisent à la méthode FETI [15]. D'autres conditions telles que des conditions de Robin [14] ou des conditions artificielles [50] peuvent également être utilisées. Ces conditions permettent de traiter les problèmes non-elliptiques.

3.4 Conclusion

Dans le cas des simulations numériques réalistes intègrant un grand nombre de paramètres, les systèmes d'équations linéaires que nous obtenons sont souvent de grande taille et ne peuvent être résolus que sur des calculateurs parallèles. Généralement, les matrices de ces systèmes ne présentent pas une structure particulière qui permettrait d'exploiter efficacement ces machines. Dès lors, les volumes de communications importants générés par les méthodes de résolution classiques, qu'elles soient directes ou itératives, conduisent à des résultat souvent décevants.

Les méthodes additives de Schwarz ont été développées par l'analyse numérique pour résoudre ce problème. Ces méthodes sont construites à partir d'un mélange d'algèbre linéaire et de certaines propriétés mathématiques des opérateurs différentiels concernés. Leur principe est de décomposer l'espace d'approximation en sous-espaces, de calculer puis de corriger une solution globale approchée sur chacun de ces sous-espaces [41]. Ce principe permet de construire des matrices par blocs, structure parfaitement adaptée aux architectures parallèles, ainsi que des préconditionneurs associés. Les méthodes additives ne sont qu'une partie des méthodes de Schwarz qui comprennent également les méthodes dites mutliplicatives. La nature des ces dernières est séquentielle. Cependant, de nombreux préconditionneurs additifs intègrent un aspect multiplicatif. Le

principe de décomposition et de correction permet de construire un cadre abstrait pour établir la convergence de ces méthodes [65]. En électromagnétisme, la décomposition de Helmohltz est utilisée pour résoudre le problème de non-ellipticité posé par les formulations magnétodynamiques, les formulations magétostatiques étant, quant à elles, des formulations elliptiques du second ordre. Les travaux récents sur ce sujet montrent que les méthodes de Schwarz (additives et multiplicatives) permettent de traiter ces formulations de manière efficace. Le principe de décomposition et de correction ainsi que le cadre abstrait et la décomposition de Helmohltz ont été présentés dans la section 3.2.

Les méthodes additives de Schwarz permettent toutes d'exploiter efficacement les machines parallèles. Il est cependant difficile de les comparer entre elles, tant leur efficacité ou leur inefficacité dépendent du problème traité. Ainsi, les méthodes multi-niveaux sont conseillées pour des équations aux dérivées partielles elliptiques dont les coefficients ne présentent pas de discontinuités et qui sont traitées sur des maillages quasi-uniformes. Lorsque ces coefficients présentent des discontinuités, il est préférable d'utiliser les méthodes de sous-domaines car le préconditionnement par blocs permet de circonscrire l'instabilité liée aux sauts de coefficients. Ces méthodes sont également conseillées lorsque les maillages ne sont pas quasi-uniformes. Le choix entre méthodes avec recouvrement et méthodes sans recouvrement n'est pas facile. Les méthodes avec recouvrement font des calculs redondants, mais le recouvrement permet d'accélérer la convergence par rapport aux méthodes sans recouvrement. Elles sont cependant réservées aux problèmes elliptiques du second ordre. A l'inverse, les méthodes sans recouvrement permettent de traiter d'autres types de problèmes. Elles présentent un meilleur parallélisme puisque les échanges d'informations entre les sous-domaines ne concernent que les inconnues des interfaces. Cependant, leur adaptation à la dimension trois est plus délicate que celle des méthodes avec recouvrement. La section 3.3 a présenté les principes généraux de ces méthodes ainsi que leurs limitations.

Deuxième partie

Contribution

Chapitre 1

Approche parallèle légère

1.1 Introduction

L'application qui nous intéresse est un code Fortran 77 développé par le Laboratoire d'Electrotechnique et d'Electronique de Puissance de Lille (L2EP) [49]. Il utilise les éléments finis de Whitney [9] pour modéliser des problèmes magnétostatiques et magnétodynamiques en trois dimensions. La section 1.2 présente son algorithme général ainsi que ses caractéristiques principales.

La parallélisation d'un code suppose des choix en matière de modèle d'exécution, de modèle de programmation ainsi qu'en outils de développement. Ces choix sont arrêtés en fonction de certains critères et forment généralement le meilleur compromis possible au vu de ces derniers. Parmi ces critères citons :

- les caractéristiques de l'application qui concernent aussi bien les types de données manipulées et les algorithmes utilisés, que la nature même de l'application (application industrielle figée ou très peu modifiée, application de recherche en constante évolution, etc.) ;
- les critères classiques du génie logiciel tels que l'efficacité, la portabilité, la facilité de développement et de maintenance, ainsi que la capacité d'évolution ;
- le public concerné (informaticien de formation ou non).

Dans la section 1.3, nous présentons les critères qui sont les nôtres et nos choix de programmation, en expliquant pourquoi ils forment à nos yeux le meilleur compromis possible. Ces choix sont respectivement le langage à parallélisme de données standard High Performance Fortran (HPF) [38], les compilateurs HPF étant libres dans le choix du modèle d'exécution, la bibliothèque de communication Halos [3], spécialisée dans les applications irrégulières basées sur le parallèlisme de données et le système de compilation HPF ADAPTOR [1].

Par approche parallèle légère, nous entendons la parallélisation d'un code existant sans remise en cause de sa structure générale et de ses algorithmes fondamentaux, la seule opération que nous autorisions étant l'ajout, en très faible quantité, de directives HPF ou de lignes de code. Bien que nous sachions que les meilleures versions parallèles des codes éléments finis sont basées sur les méthodes de Schwarz, les objectifs d'une telle démarche sont multiples :

- rendre la programmation parallèle abordable pour des non-spécialistes via

l'utilisation d'un langage de haut niveau tel que HPF, tout en garantissant un minimum de performances et une forte capacité de maintenance et d'extension pour l'application résultante ;

- valider le couplage entre le modèle à parallélisme de données incarné par HPF et les bibliothèques de communication qui sont l'apanage du modèle à parallélisme de tâches, pour une application de grande envergure ;
- se familiariser suffisamment avec l'application pour prévoir les changements de structure et d'algorithmes nécessaires à l'introduction des méthodes de Schwarz et réunir suffisamment d'informations permettant de justifier de l'emploi d'une méthode en particulier.

Les versions magnétostatiques et magnétodynamiques du code ne différant principalement que par l'utilisation d'une boucle de temps, les techniques de parallélisation employées sont valides pour ces deux versions. Nous nous sommes donc focalisés sur la parallélisation de la partie magnétostatique du code. La section 1.4, présente la démarche de parallélisation employée pour cette partie, ainsi que les résultats obtenus.

Cette approche parallèle légère a donné lieu à de nombreuses publications, les plus importantes étant [24, 25].

1.2 Le code

L'application que nous devons paralléliser a été développée par le Laboratoire d'Electrotechnique et d'Electronique de Puissance de Lille (L2EP). Il s'agit d'un code éléments finis 3D, écrit en Fortran 77, qui utilise les formulations présentées dans la partie I pour modéliser les problèmes magnétostatiques et magnétodynamiques. La figure 1.1 présente la structure générale de son algorithme.

Le maillage ne contient que des tétraèdres. Une fois ce dernier réalisé, le problème à traiter doit être décrit. Cette description consiste à donner les caractéristiques physiques des matériaux employés, les vecteurs et les normes (en coordonnées cartésiennes ou polaires) relatifs aux inducteurs et aux aimants, ainsi que des conditions aux limites homogènes de Dirichlet.

Le mailleur ne fournissant que des nœuds et des éléments, les arêtes et les facettes doivent être calculées.

La grande particularité de ce code de calcul est que les formulations sont rendues compatibles au sens des éléments de Whitney. Dans ces conditions, il n'est pas nécessaire d'imposer des conditions de jauge [54]. Cette compatibilité implique la discrétisation des termes sources représentés par les inducteurs (densité de courant $\mathbf{j}$ et champ source $\mathbf{h_s}$). La densité de courant doit être discrétisée dans l'espace des éléments finis de facettes et être à divergence nulle, tandis que le champ source $\mathbf{h_s}$ doit être discrétisé dans l'espace des éléments finis d'arêtes. Les techniques employées sont respectivement celle de l'arbre de facettes qui utilise la loi de conservation du flux dans les éléments et celle de l'arbre d'arêtes qui utilise le thérorème d'Ampère sur les facettes [48].

La discrétisation du temps est basée sur la méthode implicite d'Euler. Deux méthodes de semi-linéarisation sont employées pour traiter les systèmes d'équa-

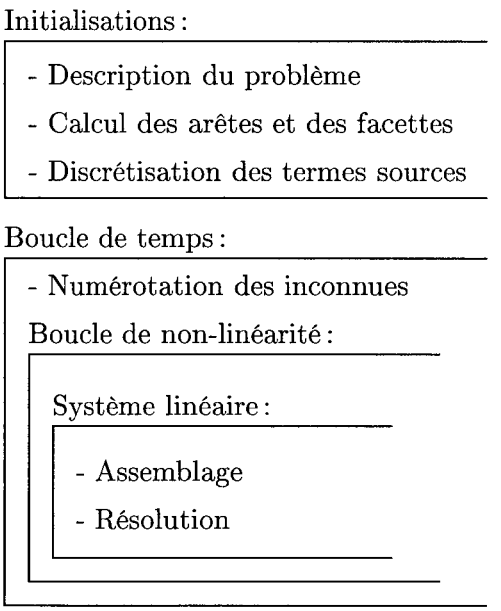

Fig. 1.1 – Algorithme général

tions non-linéaires : la méthode de Newton-Raphson et la méthode du point fixe. Leur utilisation dépend de la formulation employée.

Les inconnues numérotées, l'assemblage du système d'équations peut commencer. Les matrices obtenues étant creuses et symétriques, seule la partie triangulaire inférieure est explicitement stockée et compressée au format CSR (Compressed Sparse Row), encore appelé format morse. Ces matrices sont définies ou semi-définies positives. L'absence de condition de jauge impose l'utilisation de la méthode gradient conjugué. Il s'agit d'une méthode itérative qu'il faut préconditionner. La factorisation incomplète de Cholesky est le préconditionneur le plus couramment employé. Cependant, cette méthode ne s'adresse qu'aux matrices symétriques et définies positives. Les nôtres pouvant être semi-définies positives, nous devons employer une méthode plus générale, en l'occurrence une factorisation incomplète de Crout.

Dans cette section, nous commençons par présenter les techniques utilisées pour discrétiser les termes sources relatifs aux inducteurs. Il s'agit d'algorithmes de graphes appliqués aux arêtes et aux facettes du maillage. Nous présentons ensuite les principales caractéristiques de la phase d'assemblage et notamment le format de compression employé pour stocker les matrices. Enfin, nous terminons en présentant la méthode du gradient conjugué préconditionné que nous utilisons pour résoudre les systèmes d'équations.

1.2.1 Discrétisation des termes sources

Le problèmes posé est de satisfaire les équations :

$$\mathbf{div}\,\mathbf{j} = 0 \tag{1.2.1}$$

$$\mathbf{rot}\,\mathbf{h} = \mathbf{j} \tag{1.2.2}$$

au niveau discret. Cette satisfaction passe par la discrétisation des termes sources représentés par les inducteurs. Les deux outils utilisés sont la loi de conservation du flux et le théorème d'Ampère.

Densité de courant

Nous ne considérons ici que les inducteurs filaires à section constante. Pour satisfaire l'équation 1.2.1, nous allons discrétiser la densité de courant $\mathbf{j}$ dans l'espace $\mathbf{W^2}$ des éléments finis de facettes et appliquer la loi de conservation du flux dans chaque élément composant l'inducteur. Cette densité est donnée sous forme analytique dans la description du problème à traiter.

La discrétisation de $\mathbf{j}$ peut être réalisée en construisant un arbre de facettes. De manière informelle, il s'agit de rechercher les facettes sur lesquelles le flux de $\mathbf{j}$ doit être calculé (arbre) et les facettes sur lesquelles ce flux peut être fixé de manière arbitraire (co-arbre). Les facettes du co-arbre sont utilisées pour assurer la conservation du flux dans chaque élément composant l'inducteur. Ainsi, nous pouvons calculer le flux de $\mathbf{j}$ sur trois facettes d'un tétraèdre et utiliser la dernière pour assurer la conservation du flux dans ce tétraèdre.

Toutes les facettes externes de l'inducteur, sauf une, appartiennent à l'arbre. L'unique facette externe du co-arbre est utilisée comme point d'entrée dans l'inducteur. Certaines facettes internes sont ensuite choisies de manière à établir un chemin permettant de passer une et une seule fois par tous les éléments constituant l'inducteur. Ce chemin ne comporte donc pas de cycles, d'où le nom d'arbre. A l'issue de cette construction, il existe un unique chemin reliant deux éléments de l'inducteur. La figure 1.2 représente l'arbre et le co-arbre (en gras) de facettes d'un inducteur. La flèche représente le chemin permettant de passer une et une seule fois par tous les éléments, via les facettes du co-arbre.

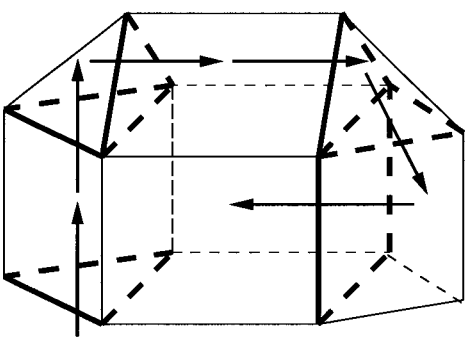

Fig. 1.2 – Arbre et co-arbre (en gras) de facettes

Champ source

Pour satisfaire l'équation 1.2.2, nous allons discrétiser le champ source $\mathbf{h_s}$ dans l'espace $\mathbf{W^1}$ des éléments finis d'arêtes et appliquer le théorème d'Ampère sur toutes les facettes du maillage. Nous supposerons ici que la densité de courant

a été discrétisée dans l'espace $\mathbf{W^2}$ des éléments finis de facettes et qu'elle est à divergence nulle.

La discrétisation de $\mathbf{h_s}$ peut être réalisée en construisant un arbre d'arêtes. De manière informelle, il s'agit de rechercher les arêtes sur lesquelles la circulation de $\mathbf{h_s}$ doit être calculée (co-arbre) et les arêtes sur lesquelles cette circulation peut être fixée de manière arbitraire (arbre). Les arêtes de l'arbre sont utilisées pour satisfaire le théorème d'Ampère sur toutes les facettes du maillage. La figure 1.3 présente l'application de ce théorème pour une facette triangulaire. Dans cette figure, les vecteurs $\mathbf{j}$ et $\mathbf{n}$ désignent respectivement la densité de courant et la normale à la facette (n_1, n_2, n_3). D'après le théorème d'Ampère, la somme des circulations $(\mathbf{h_1}, \mathbf{h_2}, \mathbf{h_3})$ de $\mathbf{h_s}$ sur les arêtes orientées de la facette est égale au flux de $\mathbf{j}$ à travers cette facette.

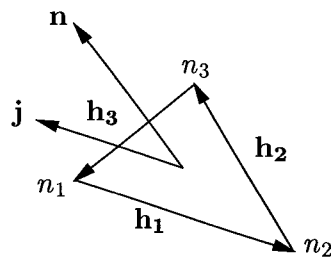

Fig. 1.3 – Théorème d'Ampère

Les arêtes auxquelles sont associées des conditions homogènes de Dirichlet font systèmatiquement partie de l'arbre. Leur valeur est déjà connue. D'autres arêtes sont ensuite choisies de manière à établir un chemin reliant tous les nœuds du maillage. Ce chemin ne comporte pas de cycles, d'où le nom d'arbre. A l'issue de cette construction, il existe un unique chemin permettant de relier deux nœuds du maillage. La figure 1.4 présente l'arbre (en gras) et le co-arbre d'arêtes d'un maillage.

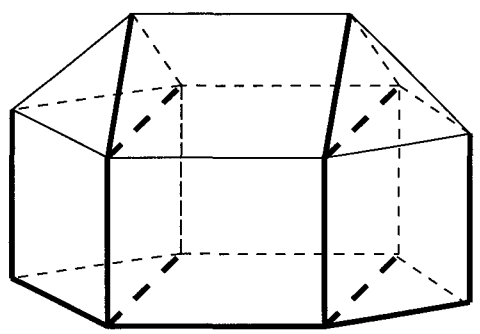

Fig. 1.4 – Arbre (en gras) et co-arbre d'arêtes

1.2.2 Assemblage

Comme nous l'avons dit dans la partie I, le système d'équations peut être construit élément par élément en calculant la contribution de chaque élément et en sommant toutes ces contributions pour obtenir le système final. Les matrices des systèmes d'équations que nous obtenons sont symétriques et creuses. Le caractère creux est lié à la numérotation des inconnues du problème, ces inconnues pouvant être relatives aux nœuds, aux arêtes, ou bien encore aux nœuds et aux arêtes du maillage. Dans ces matrices, une entrée (i, j) est non-nulle si et seulement si les inconnues i et j appartiennent à au moins un même élément du maillage et si aucune condition à la limite homogène de Dirichlet n'est associée à l'une de ces inconnues. Ces matrices pouvant avoir des tailles très importantes, il n'est bien sûr pas question de les stocker telles quelles. Seule est conservée la partie triangulaire inférieure que nous compressons afin de ne stocker que ses entrées non-nulles. Le format de compression utilisé est le format CSR (Compressed Sparse Row), encore appelé format morse. La figure 1.6 en présente le principe général. Ce format nécessite trois tableaux unidimensionnels. Les tableaux A et jA sont respectivement utilisés pour stocker les entrées non-nulles et les indices de colonnes correspondants. Le tableau iA est utilisé pour stocker les indices dans les tableaux A et jA des premières entrées non-nulles de chaque ligne. Ainsi, supposons un indice $k \in [iA(i), iA(i+1)[$. L'indice $j = jA(k)$ est alors l'indice de colonne de l'entrée non-nulle $A(k)$. Dès lors, si M désigne la matrice à compresser, nous avons $M(i, j) = A(k)$. Par la suite, nous désignerons les tableaux iA et jA sous le nom de structure CSR. La construction de cette structure est effectuée après la numérotation des inconnues, juste avant la phase d'assemblage du système d'équations.

Le mécanisme d'assemblage est illustré par la figure 1.5.

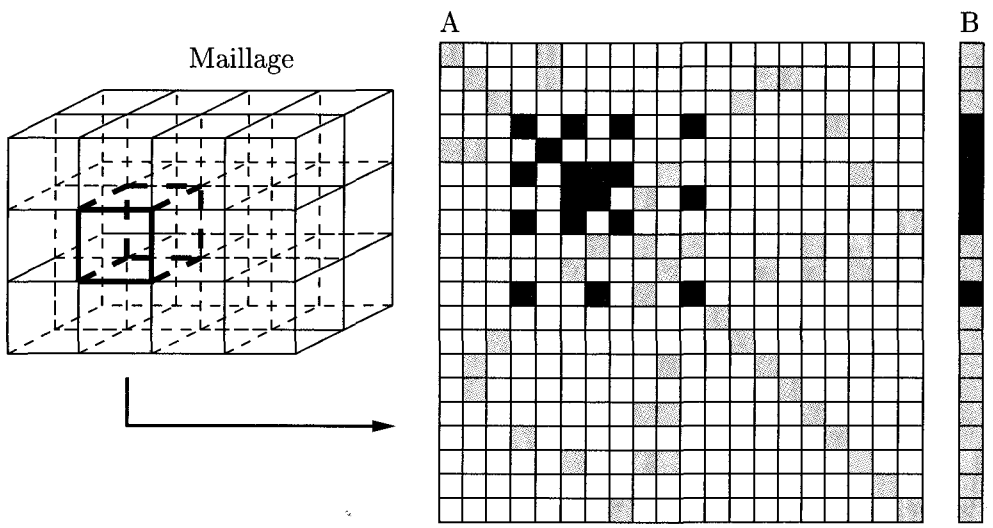

Fig. 1.5 – Assemblage du système d'équations

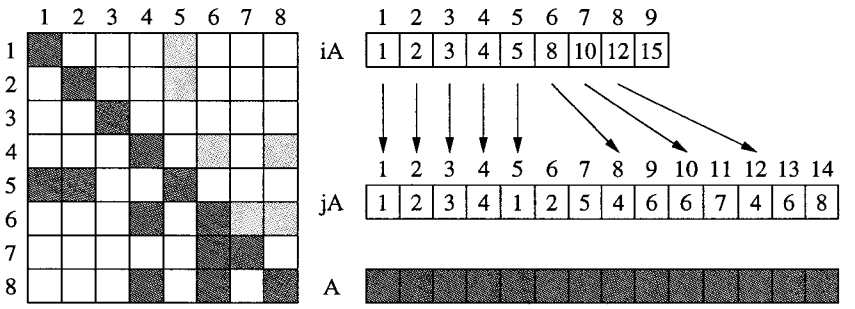

Fig. 1.6 – Format de compression CSR

1.2.3 Résolution

Le fait de ne pas jauger le problème impose que la méthode de résolution de nos systèmes d'équations soit celle du gradient conjugué. Pour le préconditionner, nous utilisons une factorisation incomplète de Crout qui permet de traiter les matrices symétriques et définies ou semi-définies positives.

Gradient conjugué préconditionné

Soit $Ax = b$ un système d'équations linéaire où A est une matrice $n \times n$ symétrique et définie positive. La solution de ce système est le point où la fonctionnelle J, définie sur $\mathbb{R}^n$ par [46] :

$$J(x) = \frac{1}{2}(Ax, x) - (b, x) \tag{1.2.3}$$

atteint son minimum. La méthode du gradient conjugué permet d'approcher ce minimum. Il s'agit d'une méthode itérative appartenant à la famille des méthodes dites de Krylov.

Soient x^0 un vecteur quelconque de $\mathbb{R}^n$ et $r^0 = b - Ax^0$ le résidu initial supposé non-nul (dans le cas contraire, le problème est résolu). Nous posons :

$$h^0 = r^0 \quad et \quad g^0 = h^0 \tag{1.2.4}$$

Le résidu initial étant non-nul, nous définissons le nombre :

$$\rho_0 = \frac{(g^0, h^0)}{(h^0, Ah^0)} \tag{1.2.5}$$

puis les vecteurs :

$$x^1 = x^0 + \rho_0 h^0 \tag{1.2.6}$$

$$g^1 = g^0 - \rho_0 Ah^0 \tag{1.2.7}$$

Soit x^p la solution approchée à l'étape p. Nous supposons construits de proche en proche les vecteurs $x^1, g^1, h^1, ..., x^{p-1}, g^{p-1}, h^{p-1}$ et x^p, g^p. Si g^p est

nul, alors x^p est solution du système. Dans le cas contraire, nous définissons le vecteur :

$$h^p = g^p + \frac{(g^p, g^p)}{(g^{p-1}, g^{p-1})} h^{p-1} \tag{1.2.8}$$

puis le nombre :

$$\rho_p = \frac{(g^p, h^p)}{(h^p, Ah^p)} \tag{1.2.9}$$

et enfin les vecteurs :

$$x^{p+1} = x^p + \rho_p h^p \tag{1.2.10}$$
$$g^{p+1} = g^p - \rho_p Ah^p \tag{1.2.11}$$

Cette méthode permet de construire une suite de directions de descente orthogonales par rapport au produit scalaire défini par A :

$$(h^j, Ah^i) = 0, \quad 0 \leq i < j \leq n \tag{1.2.12}$$

De plus, les directions g^j sont orthogonales deux à deux et orthogonales aux directions h^i:

$$(g^j, g^i) = 0, \quad 0 \leq i < j \leq n \tag{1.2.13}$$
$$(g^j, h^i) = 0, \quad 0 \leq i < j \leq n \tag{1.2.14}$$

Il s'agit d'un procédé d'orthogonalisation de Graham-Schmidt. Supposons les vecteur g^i non-nuls. La solution approchée x^p réalise le minimum de la fonctionnelle J définie sur l'espace affine $x^0 + K^p$ où K^p est l'espace de Krylov de dimension p, engendré par la base $(g^0, ..., g^{p-1})$. Nous montrons que la base $(h^0, ..., h^{p-1})$ est également une base de cet espace. La méthode dégénère si $g^p = 0$ ou si $h^p = 0$. Dans le premier cas, x^p est solution du système. Dans le second cas, nous montrons que h^p ne s'annule que si g^p s'annule. Il en résulte que la dégénérescence de la méthode signifie sa convergence. De plus, cette méthode converge en au plus n itérations, puisqu'il ne peut y avoir au plus que n vecteurs g^i orthogonaux entre-eux, ceux-ci formant une famille libre de $\mathbb{R}^n$.

Si la matrice symétrique A est seulement semi-définie positive et si b est dans l'image de A, alors si nous prenons un vecteur x^0 de $\mathbb{R}^n$ appartenant à l'image de A, la méthode converge vers l'unique solution x' du système vérifiant : $x' \in im(A)$.

La matrice A étant symétrique, définie ou semi-définie positive, son conditionnement est égal au rapport de sa plus grande et de sa plus petite (en modules) valeur propre. Pour accélérer la convergence du gradient conjugué, nous devons chercher une matrice C, spectralement proche de l'inverse de A et résoudre le nouveau système (préconditionné à gauche) $CAx = Cb$. Les préconditionneurs C les plus couramment employés sont la factorisation incomplète de Cholesky et le préconditionneur SSOR. Dans notre cas, nous avons choisi une factorisation incomplète de Crout. Il s'agit d'une factorisation de type LDL^T

permettant de traiter les matrices symétriques définies ou semi-définies positives. Cette factorisation possède la même structure que A, c'est à dire que nous ne calculons l'entrée $C(i, j)$ que si l'entrée $A(i, j)$ est non-nulle.

L'algorithme général du gradient conjugé préconditionné est donné par :

- calculer $r_0 = b - Ax^0$, $z^0 = C^{-1}r^0$, $p^0 = z^0$
- pour j = 0, 1, 2, ... jusqu'à convergence :
  - $\alpha_j = \frac{(r^j, z^j)}{(Ap^j, p^j)}$
  - $x^{j+1} = x^j + \alpha_j p^j$
  - $r^{j+1} = r^j - \alpha_j Ap^j$
  - $z^{j+1} = C^{-1}r^{j+1}$
  - $\beta_j = \frac{(r^{j+1}, z^{j+1})}{(r^j, z^j)}$
  - $p^{j+1} = z^{j+1} + \beta_j p^j$

1.3 Choix de programmation

Pour trouver les meilleurs choix de programmation, nous sommes partis d'un ensemble de contraintes :

1. le public concerné est composé d'électrotechniciens qui utilisent Fortran 77 ;
2. le code de simulation utilisé est un code de recherche. Il est constamment développé et modifié afin de tester de nouvelles formulations ou d'intégrer de nouveaux paramètres de simulation tels que le comportement des matériaux ;
3. il s'agit d'un code irrégulier dans la mesure où les matrices manipulées sont creuses et que leur accès s'effectue de manière indirecte par l'intermédiaire des indices de la structure CSR, ces indices n'étant connus qu'au moment de l'exécution.

Si nous traduisons ces contraintes dans le domaine informatique, nous obtenons :

1. le langage de programmation à utiliser est Fortran ;
2. pour des codes de recherche en constante mutation, la facilité de maintenance et la capacité d'évolution sont des caractéristiques fondamentales. Parce qu'il laisse au compilateur le soin d'adapter le code à la machine cible, le modèle à parallélisme de données présente ces caractéristiques ;
3. la structure CSR du code est calculée durant son exécution. Elle n'est donc pas disponible lors de la compilation. Comme nous l'avons dit dans la partie I, l'efficacité d'un code basé sur le parallélisme de données dépend fortement du volume d'informations disponible à la compilation. Dans notre cas, ignorant le contenu de la structure CSR, le compilateur ne pourra pas établir la carte des communications générées par le code durant la phase de compilation. Cette carte est utilisée pour optimiser les communications inter-processeurs. Elle devra donc être déterminée durant l'exécution, pour chaque communication, ce qui dégradera d'autant les performances du code. Dans ces conditions, le recours aux bibliothèques de communication

est inévitable. Or, de telles bibliothèques qui permettent de gérer manuellement les communications, sont le propre du modèle à parallélisme de tâches.

Ces constatations nous amènent à dire que l'application parallèle à réaliser sera basée sur le parallélisme de données, mais que pour des raisons d'efficacité, certaines communications devront être explicites et utiliser des bibliothèques de communication. Notons qu'une telle approche, aussi efficace soit-elle, n'en constitue pas moins une violation des principes du modèle à parallélisme de données. D'autres approches, plus en accord avec ces principes, ont également été envisagées. Citons par exemple une approche basée sur une représentation hiérarchisée des structures de données irrégulières et devant permettre une prise en charge plus efficace de cette irrégularité par le compilateur [20, 22, 23]. Cependant, en raison de la complexité des applications de simulation numériques qui sont des applications complètes et non des noyaux de codes extraits de leur contexte d'origine, des contraintes d'efficacité imposées à ces applications et du caractère expérimental d'une telle approche, nous ne l'avons pas retenue.

Les outils qui peuvent nous permettre de réaliser une telle application sont :

1. Fortran 90 est une norme internationale depuis 1991 [45]. Ce langage est le successeur de Fortran 77. Il se hisse au niveau des grands langages procéduraux tels que C ou Ada, grâce à l'introduction des modules, des types définis par le programmeur, de l'allocation dynamique, de la récursivité, du calcul vectoriel et du contrôle de la précision numérique. Fortran 95 est une évolution mineure de Fortran 90, consécutive aux travaux menés sur le projet HPF (High Performance Fortran) [38]. Les améliorations apportées concernent essentiellement l'algorithmique vectorielle. Fortran 90 et Fortran 95 permettent de supprimer les caractéristiques les plus discutables de Fortran 77 ;
2. HPF est le langage de programmation à parallélisme de données standard. L'objectif du projet HPF était de définir un langage de programmation pour les calculateurs parallèles qui serait compatible avec Fortran, suffisamment général pour s'adapter à différentes plateformes et qui offrirait à l'utilisateur des fonctionnalités simples à utiliser pour obtenir un minimum de performance quelle que soit la machine cible [44]. La version 1.0 de ce langage, officiellement distribuée à partir de 1993, est définie comme un Fortran 90 augmenté de directives permettant de spécifier la répartition des données et la manière de les exploiter. Ces directives ne sont que des "conseils" adressés au compilateur, ce dernier étant libre de les appliquer ou de les rejeter. Dans cette version, le compilateur est libre du choix du modèle d'exécution bien que le modèle SPMD soit le plus approprié. Il est également responsable de la mise en place de toutes les communications nécessaires, ces dernières étant implicites pour l'utilisateur. Comme HPF intègre la notion de répartition des données dans son modèle de programmation, il est parfaitement adapté aux machines parallèles à mémoire répartie. Cependant, le compilateur étant libre du choix du modèle d'exécution, un même code HPF peut être exécuté sur différents types de plateformes. Notons que les codes HPF ne peuvent s'exécuter que sur un

ensemble de processeurs homogènes. La version 2.0 est apparue en 1997. Elle précise la première version et définit une liste d'extensions (optionnelles) approuvées et destinées, d'une part, à uniformiser l'implémentation des compilateurs et, d'autre part, à assurer une meilleure prise en compte des applications irrégulières par des directives et des clauses permettant au programmeur de donner des informations au compilateur que ce dernier ne peut découvrir. Certaines de ces extensions seront détaillées par la suite ;

3. les bibliothèques de communication les plus couramment utilisées par les applications basées sur le parallélisme de tâches sont PVM et MPI [35, 36]. Ces dernières proposent un ensemble de primitives de bas niveau nécessaires à la communication et à la synchronisation de tâches sur un ensemble de processeurs, cet ensemble pouvant être homogène ou hétérogène. Ainsi, la communication entre tâches utilise un schéma stockage-encodage, envoi-réception, déstockage-décodage. Le stockage consiste à déposer les données en partance dans une zone tampon. Si les processeurs sont hétérogènes, ces données doivent être codées dans un format universellement admis (format XDR). La tâche qui reçoit ces données, doit, quant à elle, effectuer les opération inverses.

   Des bibliothèques de communication pour les applications irrégulières basées sur le parallélisme de données ont également été développées. Ces bibliothèques sont nées du projet HPF+ [3] et se sont inspirées du principe des mémoires "shadow". Les mémoires "shadow" sont une extension approuvée de HPF v2.0 qui permettent de représenter des ensembles de données partagées entre plusieurs processeurs. Il s'agit de zones de mémoire allouées sur chaque processeur, dont le rôle est de stocker une copie de chaque donnée non-locale de l'ensemble concerné. Les communications entre les données originales et leurs copies sont déclenchées par le compilateur. Les mémoires "shadow" ont été initialement créées pour le cas régulier, la carte des communications pouvant être établie à la compilation. Leur adaptation au cas irrégulier est basée sur l'idée que la carte des communications d'une application varie peu ou pas au cours de son exécution. Il est donc possible d'établir cette carte une fois pour toute durant l'exécution puis de la réutiliser. Nous retrouvons ainsi l'efficacité du cas régulier. Ces bibliothèques peuvent être utilisées automatiquement par le compilateur. C'est notamment le cas de Vienna Fortran [5]. Un tel automatisme ayant ses limites, elles peuvent également être manipulées directement par le programmeur. Qu'elle soit automatique ou manuelle, l'utilisation de ces bibliothèques permet de conférer aux application HPF une efficacité proche de celles utilisant PVM ou MPI [18], tout en préservant une facilité de maintenance et une capacité d'évolution importante.

Dans la suite de cette section, nous présentons les caractéristiques principales de High Performance Fortran. Nous présentons ensuite Halos [4] qui est une bibliothèque de communication pour les applications irrégulières basées sur le parallélisme de données. Enfin, nous terminons en présentant ADAPTOR [1] qui est le système de compilation HPF que nous avons choisi pour réaliser notre

application. Notons que Halos est directement supportée dans ADAPTOR [18].

1.3.1 High Performance Fortran

Notre propos n'est pas de faire ici une présentation exhaustive de High Performance Fortran. Pour cela, nous renvoyons le lecteur à [38]. Nous allons présenter les caractéristiques principales de ce langage, caractéristiques que nous utiliserons par la suite, en nous appuyant sur une série de petits exemples. Pour chaque exemple, nous présentons deux versions : l'une écrite en Fortran 90 et l'autre en HPF.

Ajout de directives

Considérons l'exemple présenté dans la figure 1.7. Il s'agit d'un exemple bien connu dans la littérature HPF. Dans ce dernier, nous commençons par déclarer quatre tableaux unidimensionnels a, b, c et d contenant chacun trois entiers. Après avoir déclaré une variable de boucle i, nous recopions respectivement les contenus des tableaux b et c dans les tableaux a et d. Notons que dans la réalité, Fortran 90 disposant d'un jeu d'instructions vectorielles, nous aurions tout simplement écrit $a = c$ et $b = d$.

```
      program exemple_1_f90
        integer, dimension(1:3) :: a, b, c, d
        integer :: i

        do i = 1, 3
          a(i) = b(i)
          c(i) = d(i)
        end do
      end program exemple_1_f90
```

```
      program exemple_1_hpf
        integer, dimension(1:3) :: a, b, c, d
        integer :: i
!hpf$ align with a :: b, c, d
!hpf$ distribute(block) :: a

!hpf$ independent, on home(a(i))
        do i = 1, 3
          a(i) = b(i)
          d(i) = c(i)
        end do
      end program exemple_1_hpf
```

Fig. 1.7 – Parallélisation HPF, exemple 1

Cette figure permet d'illustrer le principe de HPF. Ce principe consiste à prendre un programme séquentiel et à y ajouter des directives permettant de spécifier la répartition des données et la manière de les exploiter. Nous remarquons que ces directives sont précédées du caractère annonçant une ligne de

commentaires en Fortran 90 (ce caractère est également interprété comme tel par la plupart des compilateurs Fortran 77 actuels). Dès lors, elles ne sont interprétables que par un compilateur HPF, les compilateurs Fortran 90 les assimilant à des lignes de commentaires. Les instructions étant identiques, un même code peut donc être exécuté sur un machine séquentielle et sur une machine parallèle, la seule opération nécessaire étant une recompilation de ce dernier.

Répartition des données

Les directives concernant les données sont ajoutées dans la partie déclarative du programme. Dans l'exemple de la figure 1.7, la directive `align` indique au compilateur que les éléments de tableaux $a(i)$, $b(i)$, $c(i)$ et $d(i)$ seront attribués à un même processeur. Il ne s'agit pas d'une directive de répartition des données mais d'une information indiquant au compilateur que lorsque celui-ci rencontrera une directive de répartition des données concernant le tableau a, il devra répartir les données des tableaux b, c et d de la même manière. La directive `distribute` est une directive de répartition des données. Le paramètre `block` passé à cette directive indique au compilateur qu'il doit affecter une tranche de N/P éléments consécutifs à chaque processeur, N désignant le nombre d'éléments du tableau et P le nombre de processeurs utilisés pour exécuter le code. Notons que si le reste de cette division n'est pas nul, le nombre d'éléments affectés aux $P-1$ premiers processeurs est $N/P+1$, le dernier processeur se voyant attribuer les éléments restants. Il est possible de passer d'autres paramètres à cette directive. Ainsi, le programmeur peut décider lui-même du nombre d'éléments à affecter à chaque processeur. Lorsque ce nombre varie selon les processeurs, nous parlons de distribution par blocs généralisée. Le paramètre utilisé est alors `gen_block` suivi d'un tableau unidimensionnel contenant les nombres d'éléments à affecter à chaque processeur. La taille de ce tableau donne le nombre de processeurs nécessaires à l'exécution du code. La distribution par blocs généralisée est une extension approuvée de HPF v2.0.

La répartition des données peut être modifiée durant l'exécution du programme. Cette modification peut être implicite ou explicite. Elle est implicite lorsque certaines fonctions exigent que leurs paramètres soient répartis d'une certaine manière. Dans ce cas, si la répartition d'un tableau dans le programme appelant est différente de la répartition du paramètre correspondant dans la fonction appelée, la modification est implicitement gérée par le compilateur. Elle est explicite lorsqu'elle est demandée par le programmeur. Dans ce cas, les directives de réalignement et de re-répartition dynamiques `realign` et `redistribute` sont ajoutées dans la partie instructions du programme. Les tableaux concernés par ces directives dynamiques doivent comporter l'information `dynamic` dans leur déclaration.

Notons que les variables scalaires sont systématiquement répliquées sur l'ensemble des processeurs. C'est le cas de la variable de boucle i dans l'exemple de la figure 1.7. Il en est de même pour les tableaux ne faisant l'objet d'aucune directive de répartition des données.

Contrôle du code

Les directives concernant le code sont ajoutées dans la partie instructions du programme. Dans l'exemple de la figure 1.7, la directive `independent` indique au compilateur que les itérations de la boucle peuvent être exécutées dans n'importe quel ordre, y compris en parallèle. La clause `on home` lui indique que la ième itération de cette boucle sera effectuée par le processeur possédant l'élément de tableau $a(i)$. Quand elle n'est pas présente, le compilateur applique la règle du "propriétaire calcule" (Owner Compute Rule). En toute rigueur, cette clause n'est pas utile dans l'exemple proposé puisque l'information apportée au compilateur par la directive `align` est suffisante pour lui permettre de répartir les itérations de la boucle. Cette information lui permet également de savoir que chaque processeur dispose des données dont il a besoin pour assurer ses propres itérations de la boucle. Il n'y a donc pas de communications inter-processeurs à mettre en place. La figure 1.8 présente respectivement l'ordre d'exécution des lectures et des écritures dans les boucles des versions Fortran 90 et HPF.

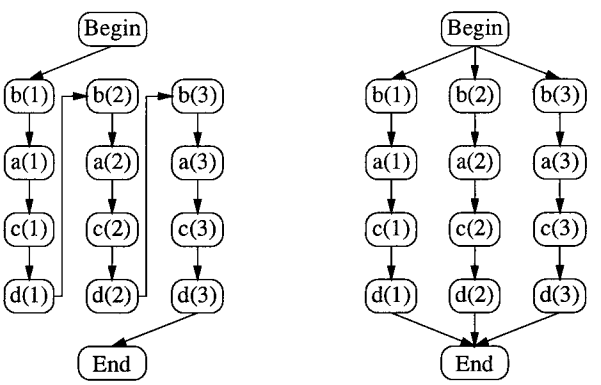

Fig. 1.8 – Ordre d'exécution des boucles (Fortran 90, HPF)

Il est possible d'aligner des tableaux de tailles et de dimensions différentes mais cette opération demande parfois une certaine expertise de la part du programmeur. En outre, de tels alignements doivent disposer d'une justification algorithmique. Par la suite, par soucis de simplicité, nous supposerons que seuls des tableaux de tailles et de dimensions identiques peuvent être alignés.

Considérons à présent l'exemple de la figure 1.9. Dans cet exemple, nous déclarons deux tableaux unidimensionnels a et b contenant chacun trois entiers. Nous déclarons également un autre tableau unidimensionnel c mais contenant six entiers. Le rôle de la première boucle est de calculer la somme des éléments $c((i-1)*2+1)$ et $c((i-1)*2+2)$ et de la stocker dans $a(i)$. Le rôle de la seconde boucle est de calculer le produit scalaire du vecteur b par lui-même.

Les tableaux a et c ayant des tailles différentes, nous considérons qu'il n'est pas possible de les aligner l'un sur l'autre. Nous choisissons de leur appliquer une directive de répartition par blocs généralisée. Ainsi, chaque processeur reçoit un éléments du tableau a et du template t ainsi que deux éléments du tableau c. Le nombre de processeurs nécessaire pour exécuter le code doit alors être supérieur ou égal à trois. Contrairement à l'exemple précédent, nous avons omis

```
      program exemple_2_f90
        integer, dimension(1:3) :: a, b
        integer, dimension(1:6) :: c
        integer :: i, j, k, s

        do i = 1, 3
          j = (i - 1) * 2 + 1
          k = j + 1
          a(i) = c(j) + c(k)
        end do

        s = 0
        do i = 1, 3
          s = s + b(i) * b(i)
        end do
      end program exemple_2_f90
```

```
      program exemple_2_hpf
        integer, dimension(1:3) :: a, b
        integer, dimension(1:6) :: c
        integer :: i, j, k, s
!hpf$ template(1:3) :: t
!hpf$ distribute(gen_block( / 1, 1, 1/ )) :: a, t
!hpf$ distribute(gen_block( / 2, 2, 2 /)) :: c

!hpf$ independent, on home(a(i)), &
!hpf$ resident(c), new(j, k)
        do i = 1, 3
          j = (i - 1) * 2 + 1
          k = j + 1
          a(i) = c(j) + c(k)
        end do

        s = 0
!hpf$ independent, on home(t(i)), &
!hpf$ reduction(s)
        do i = 1, 3
          s = s + b(i) * b(i)
        end do
      end program exemple_2_hpf
```

Fig. 1.9 – Parallélisation HPF, exemple 2

de répartir le tableau b. Dès lors, de même que pour les variables scalaires i, j, k et s, ce tableau sera répliqué sur l'ensemble des processeurs. Notons que la conduite à tenir en cas de tableau non-réparti est laissée à l'appréciation du compilateur. La réplication de ce tableau est la conduite la plus couramment rencontrée. Il est possible de la modifier via des options donnée par le programmeur au compilateur. La figure 1.10 présente le résultat de cette répartition pour 3 processeurs.

P_0	P_1	P_2
$a(1)$	$a(2)$	$a(3)$
$c(1)$, $c(2)$	$c(3)$, $c(4)$	$c(5)$, $c(6)$
$b(1)$, $b(2)$, $b(3)$	$b(1)$, $b(2)$, $b(3)$	$b(1)$, $b(2)$, $b(3)$
i, j, k, s	i, j, k, s	i, j, k, s

Fig. 1.10 – Répartition sur 3 processeurs, exemple 2

Lorsque le compilateur examine la première boucle, il ignore que les éléments $a(i)$, $c((i-1)*2+1)$ et $c((i-1)*2+2)$ appartiennent au même processeur, cette information ne lui ayant pas été explicitement indiquée. La clause `resident` permet de lui signifier que tous les processeurs disposent des données nécessaires pour assurer leurs propres itérations de la boucle et que l'exécution de cette dernière ne nécessite pas de communications inter-processeurs. Les variables j et k étant répliquées sur l'ensemble des processeurs, le compilateur se doit de garantir leur cohérence et donc de générer les communications inter-processeurs correpondantes. Or, ces variables ne sont utilisées que pour stocker des résultats temporaires et n'ont aucun rapport entre-elles d'une itération à l'autre de la boucle. Dans ces conditions, garantir leur cohérence est une perte de temps que la clause `new` permet d'éviter. Cette clause indique au compilateur que la cohérence des variables qu'elle concerne n'a pas à être assurée. Notons que si cette clause n'est pas utilisée, le compilateur refusera de paralléliser la boucle.

En regardant le code de la seconde boucle, nous sentons bien qu'il serait possible de la paralléliser en utilisant les mêmes directives que dans l'exemple de la figure 1.7. Cependant, la première difficulté qui se pose est liée au fait que le tableau b est ici répliqué. Par conséquent, nous ne pouvons pas utiliser la clause `on home` pour spécifier le lieu des itérations de la boucle. La directive `template` nous permet de résoudre ce problème. Dans HPF, un template est un faux tableau sur lequel viennent s'aligner de vrais tableaux. Son rôle est d'exprimer des alignements complexes entre tableaux de tailles ou de dimensions différentes. Lorsque ces alignements sont définis, le template fait l'objet d'une directive de répartition des données. Une autre utilisation possible du template est de répartir les itérations d'une boucle quand le tableau concerné est répliqué sur l'ensemble des processeurs. C'est l'utilisation que nous en faisons. La seconde difficulté qui se pose est liée au fait que la variable s est également répliquée sur l'ensemble des processeurs et que sa cohérence doit être garantie, contrairement aux variables j et k de la boucle précédente. La clause `reduction` permet de résoudre ce problème. Elle indique au compilateur que chaque processeur peut créer une instance locale de la variable s pour stocker le résultat correspondant

à ses propres itérations de la boucle, mais qu'à la fin de cette dernière, toutes les instances locales doivent être combinées pour obtenir le résultat final et le diffuser à tous les processeurs.

Signalons enfin que des fonctions peuvent être appelées dans les boucles parallèles. La déclaration de ces fonctions doit être précédée de l'information `pure`. Cette dernière indique au compilateur que toutes les données qui seront référencées dans cette fonction seront locales au processeur. Il existe également des extensions pour HPF qui ne font pas partie de la définition du langage. Parmi ces extensions, citons `extrinsic(hpf_serial)` qui, lorsqu'elle précède la déclaration d'une fonction, indique au compilateur que cette dernière ne sera exécutée que par un seul processeur. Si le paramètre passé à cette fonction est un tableau réparti, ce dernier sera reconstitué dans son intégralité dans la mémoire du processeur concerné. Une fois ce tableau traité, chaque processeur recevra la tranche qui lui a été attribuée par la directive de répartition.

1.3.2 Halos

Considérons l'exemple présenté dans la figure 1.11. Dans cet exemple, nous déclarons quatre tableaux unidimensionnels a, b, c et d que nous allouons dynamiquement pour contenir trois entiers. Nous déclarons également deux tableaux unidimensionnels ib et ic que nous allouons pour contenir trois indices référençant respectivement les tableaux b et c. Nous supposerons que ces indices sont tous différents et que leur valeur est comprise entre 1 et 3. Après avoir calculé cette valeur, nous l'utilisons pour recopier respectivement certaines valeurs des tableaux b et d dans les tableaux a et c. Nous désallouons ensuite tous les tableaux utilisés.

Les techniques de parallélisation employées sont identiques à celles utilisées dans les exemples précédents. Ignorant le contenu des tableaux ib et ic, le compilateur suppose l'existence de communications inter-processeurs nécessaires à l'exécution de la boucle. Pour la paralléliser, il lui faut tout d'abord amener aux processeurs les données dont ils ont besoin pour assurer leurs propres itérations de la boucle. Il lui faut ensuite ramener les résultats produits par l'exécution de cette dernière sur ces mêmes processeurs. Une stratégie habituelle est basée sur l'utilisation de tableaux temporaires. La figure 1.12 présente les modifications apportées au code du programmeur.

Le remplissage des tableaux tmp_b et tmp_c, avant et après l'exécution de la boucle, nécessite des communications inter-processeurs. Les communications relatives au tableau tmp_b sont appelées communications de pré-chargement. Celles relatives au tableau tmp_c sont appelées communications de post-stockage. L'utilisation de tableaux temporaires fait que l'exécution de la boucle ne nécessite plus de communications interprocesseurs.

La carte des communications inter-processeurs ne peut être établie qu'au moment de l'exécution de la boucle. Sa construction est basée sur un mécanisme d'inspection et d'exécution. Durant la phase d'inspection, les processeurs échangent des informations leur permettant de savoir où se trouvent les données dont ils ont besoin. Ces informations sont utilisées dans la phase d'éxécution où chaque processeur prépare, envoie et reçoit les données concernées. Les in-

```
      program exemple_3_f90
        integer, dimension(:), dynamic :: a, b, c, d, ib, ic
        integer :: i

        allocate(a(1:3), b(1:3), c(1:3), d(1:3), ib(1:3), ic(1:3))
        call calculer(ib, ic)

        do i = 1, 3
          a(i) = b(ib(i))
          c(ic(i)) = d(i)
        end do

        deallocate(a, b, c, d, ib, ic)
      end program exemple_3_f90
```

```
      program exemple_3_hpf
        integer, dimension(:), dynamic :: a, b, c, d, ib, ic
        integer :: i
!hpf$ align with a :: b, c, d, ib, ic
!hpf$ distribute(block) :: a

        allocate(a(1:3), b(1:3), c(1:3), d(1:3), ib(1:3), ic(1:3))
        call calculer(ib, ic)

!hpf$ independent, on home (a(i))
        do i = 1, 3
          a(i) = b(ib(i))
          c(ic(i)) = d(i)
        end do

        deallocate(a, b, c, d, ib, ic)
      end program exemple_3_hpf
```

Fig. 1.11 – Parallélisation HPF, exemple 3

```
      program exemple_3_hpf
        integer, dimension(:), dynamic :: a, b, c, d, ib, ic
        integer, dimension(:), dynamic :: tmp_b, tmp_c
        integer :: i
!hpf$ align with a :: b, c, d, ib, ic, tmp_b, tmp_c
!hpf$ distribute(block) :: a

        allocate(a(1:3), b(1:3), c(1:3), d(1:3), ib(1:3), ic(1:3))
        allocate(tmp_b(1:3), tmp_c(1:3))
        call calculer(ib, ic)

        forall(i = 1:3) tmp_b(i) = b(ib(i))
!hpf$ independent, on home (a(i))
        do i = 1, 3
          a(i) = tmp_b(i)
          tmp_c(i) = d(i)
        end do
        forall(i = 1:3) c(ic(i)) = tmp_c(i)

        deallocate(a, b, c, d, ib, ic)
        deallocate(tmp_b, tmp_c)
      end program exemple_3_hpf
```

Fig. 1.12 – Modifications du compilateur, exemple 3

formations collectées durant la phase d'inspection sont également utilisées pour optimiser les communications. Elles permettent à un processeur possédant les données réclamées par un autre processeur, de les regrouper et de les envoyer en une seule fois. Les communications inter-processeurs sont ainsi moins nombreuses et plus volumineuses, ce qui permet une meilleure exploitation du réseau de communication.

Le problème posé par le mécanisme d'inspection et d'exécution est son activation systématique pour chaque exécution de la boucle, ce qui dégrade d'autant les performances de l'application. Halos [4] est une bibliothèque de communication pour les applications irrégulières basées sur le parallélisme de données. Son idée de base est de constater que le contenu des tableaux d'indices de ces applications varie généralement peu ou pas au cours de l'exécution. Dès lors, tant que ce contenu ne change pas, il est possible d'effectuer une seule phase d'inspection et d'utiliser les informations collectées pour ne plus effectuer que des phases d'exécution par la suite. Tout changement de contenu des tableaux d'indices entraîne le renouvellement de ce processus. La figure 1.13 présente la répartition sur deux processeurs d'un tableau X et de son tableau d'indices associé L. Chaque processeur contient alors une partie du tableau L. Dans ce dernier, des indices référencent des entrées du vecteur X qui appartiennent à l'autre processeur. Ces indices sont appelés indices non-locaux.

La figure 1.14 illustre l'application de Halos à l'exemple de la figure 1.13.

Sur chaque processeur, Halos inspecte le tableau L pour repérer les indices non-locaux. Le nombre de ces indices détermine la taille de la mémoire "shadow"

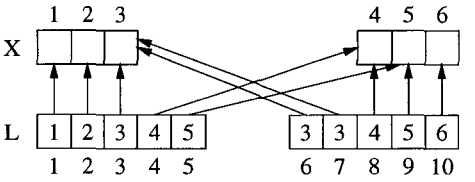

Fig. 1.13 – Indices non-locaux sur 2 processeurs

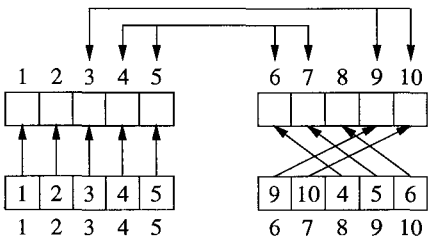

Fig. 1.14 – Halos sur 2 processeurs

à allouer sur le processeur. Notons que cette taille ne tient pas compte des répétitions d'indices. Ainsi, dans l'exemple de la figure 1.14, la mémoire "shadow" du second processeur prévoit deux emplacements pour accueillir deux copies du même élément $X(3)$. Généralement, la mémoire "shadow" est allouée sous forme d'une extension ajoutée à la fin du tableau X. Ce dernier est alors réalloué dynamiquement, c'est pourquoi sa déclaration doit comporter l'information `dynamic`. Lorsque l'extension est créée, Halos modifie les indices non-locaux du tableau L pour qu'ils référencent l'extension ajoutée au tableau X. Les informations collectées par la phase d'inspection concernent les emplacements des éléments originaux et ceux de leurs différentes copies. Ces informations restent valides jusqu'à un éventuel changement de contenu des tableaux d'indices. Durant les phases d'exécution, les communications de pré-chargement s'effectuent depuis les éléments originaux vers leurs différentes copies. Les communications de post-stockage s'effectuent depuis les copies vers les éléments originaux. Comme un même élément original peut être référencé par plusieurs indices du tableau L sur différents processeurs, ce dernier type de communication fait intervenir des opérations de réduction combinant les valeurs des copies et des éléments originaux.

La figure 1.15 présente une utilisation explicite de Halos telle qu'elle se fait dans ADAPTOR. Notons que ce compilateur permet également une utilisation implicite de cette bibliothèque.

Par rapport à l'exemple de la figure 1.11, nous déclarons deux entiers supplémentaires id_b et id_c. Ces entiers sont des identificateurs destinés à représenter les mémoires "shadow" et les informations collectées par la phase d'inspection pour les tableaux b et c. Lorsque les tableaux d'indices ib et ic sont calculés, l'appel de la routine `halo_all_define` permet d'effectuer le travail présenté dans la figure 1.14. L'appel de la routine `halo_update` exécute la communication de pré-chargement. Nous remarquons que par rapport à la pa-

```
      program exemple_3_hpf
      use hpf_halo_library
        integer, dimension(:), dynamic :: a, b, c, d, ib, ic
        integer :: i, id_b, id_c
!hpf$ align with a :: b, c, d, ib, ic
!hpf$ distribute(block) :: a

        allocate(a(1:3), b(1:3), c(1:3), d(1:3), ib(1:3), ic(1:3))
        call calculer(ib, ic)
        call halo_all_define(id_b, b, ib)
        call halo_all_define(id_c, c, ic)

        call halo_update(id_b, b)
!hpf$ independent, on home (a(i)), resident(b, c)
        do i = 1, 3
          a(i) = b(ib(i))
          c(ic(i)) = d(i)
        end do
        call halo_reduce_copy(id_c, c)

        call halo_free(id_b)
        call halo_free(id_c)
        deallocate(a, b, c, d, ib, ic)
      end program exemple_3_hpf
```

Fig. 1.15 – Parallélisation HPF-Halos dans ADAPTOR, exemple 3

rallélisation HPF de la figure 1.11, la clause `resident` a été ajoutée et concerne les tableaux b et c. Cette clause permet de supprimer la phase d'inspection et exploite le fait que les mémoires "shadow" ont été allouées sous forme d'extensions pour les tableau b et c. L'appel de la routine `halo_reduce_copy` exécute la communication de post-stockage. Il existe en fait tout un ensemble de fonctions permettant d'effectuer des réductions et cette dernière n'est qu'un représentant particulier qui peut paraître superflu. Nous verrons par la suite qu'il n'en n'est rien et que cette dernière prend tout son sens quand les données des tableaux b et c ne sont pas des valeurs numériques mais des types définis par le programmeur. Notons que la manipulation de cette fonction réclame une attention particulière car les indices du tableau L doivent être tous différents, sans quoi l'ordre des écritures devient aléatoire. C'est pourquoi nous avons bien pris soin de faire cette hypothèse dans le cas de l'exemple de la figure 1.11. L'appel de la routine `halo_free` permet de détruire les mémoires "shadow" et les informations collectées pendant la phase d'inspection.

1.3.3 ADAPTOR

ADAPTOR (Automatic DAta Parallelism translatOR) est un système de compilation HPF du domaine public, développé par le GMD-SCAI (German National Research Center for Information Technology - Institute for Algorithms and Scientific Computing) [1]. Son principe est de transformer une application HPF en application basée sur le parallélisme de tâches pour les machines à mémoire répartie, ou en application basée sur le parallélisme de tâches multithreadées pour les machines SMP et CC-NUMA. Contrairement aux applications basées sur le parallélisme de tâches ou le parallélisme de tâches multithreadées qui invoquent directement les routines des bibliothèques de communication ou de threads, les applications générées par ADAPTOR passent par l'intermédiaire d'une bibliothèque permettant de gérer la répartition des données : la DALIB (Distributed Array Library). Celle-ci est chargée d'invoquer directement les routines de la bibliothèque de communication ou de threads choisie par l'utilisateur. Le compilateur utilisé pour créer l'exécutable est le compilateur Fortran 77, 90 ou 95 de la machine utilisée. La figure 1.16 illustre le principe général d'ADAPTOR.

ADAPTOR supporte HPF v2.0 et la plupart des extensions approuvées. Il propose également des directives propriétaires qui permettent d'optimiser les performances des applications en fonction des architectures concernées. Depuis sa version 7.0 (1999), ADAPTOR supporte Halos et OpenMP. La version 8.0 (2000) est actuellement au stade de béta-version. La première amélioration apportée par cette nouvelle version concerne Halos. Elle permet de supprimer les communications multiples liées aux répétitions d'indices non-locaux, problème illustré dans la figure 1.14. La seconde amélioration est un modèle d'exécution hiérarchisé qui permet d'exploiter les machines à base de SMP qui ne sont pas CC-NUMA, ainsi que les grappes de SMP.

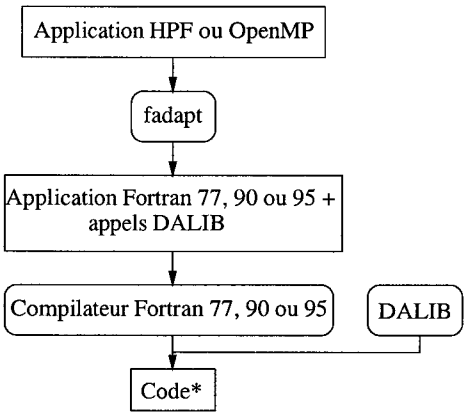

Fig. 1.16 – Système de compilation ADAPTOR

1.4 Parallélisation de la partie magnétostatique

Dans la structure générale de l'algorithme présentée en figure 1.1, nous constatons que les parties magnétostatiques et magnétodynamiques du code ne diffèrent principalement que par l'ajout d'une boucle temporelle. Il exst donc logique de penser que les techniques employées pour paralléliser la partie magnétostatique de ce code, restent valables pour sa partie magnétodynamique. Aussi nous sommes nous focalisés sur la parallélisation de la partie magnétostatique.

Il est difficile de parler de portabilité et de capacité d'extension d'une application de grande taille quand celle-ci est écrite en Fortran 77. Les problèmes posés par ce langage sont multiples. Tout d'abord, Fortran 77 autorise la non-déclaration de certaines données. Le principe du modèle à parallélisme de données étant de répartir ces dernières, il est nécessaire de les déclarer. Ensuite, les données globales d'une application sont déclarées dans le programme principal. Comme il n'est pas possible de regrouper toutes les fonctions du code dans un même fichier, ces fonctions sont dispersées dans plusieurs de ces fichiers. Elles perdent ainsi la visibilité des données globales du code. Toute manipulation de ces dernières impose alors de les passer du programme principal aux fonctions sous forme de paramètres. Aussi n'est-il pas rare de rencontrer des prototypes de fonctions avec dix, quinze voir vingt paramètres. Enfin, Fortran 77 ne prévoit ni la possibilité pour le programmeur de définir ses propres types, types garants d'une certaine cohérence des données, ni un jeu d'instructions vectorielles. Il en résulte alors une démultiplication du nombre des tableaux et une obligation de les traiter par l'intermédiaire de boucles.

Dans cette section, nous commençons par présenter une nouvelle version de la partie magnétostatique du code, version totalement ré-écrite en Fortran 90. Nous présentons ensuite une parallélisation HPF de cette nouvelle version et nous terminons en présentant les résultats obtenus pour cette dernière.

1.4.1 Version Fortran 90

Le code Fortran 77 que nous avons pris en compte a été développé par Y. Le Menach (L2EP) [48]. La partie magnétostatique a ensuite été reprise et étendue par G. Marques (L2EP) pour ses recherches sur les estimateurs d'erreurs en trois dimensions [47]. Ces deux versions, du seul point de vue informatique, souffraient de nombreuses limitations au niveau des performances. Cependant, l'ampleur de la tâche, la complexité des exemples traités et celle des techniques employées, permettent de se rendre compte de la difficulté d'élaborer de tels codes de calcul.

Dans ces versions, le seul élément géométrique utilisable dans un maillage était le tétraèdre à quatre nœuds, bien que certaines tentatives aient été faites pour introduire le prisme à six noeuds. Le temps de calcul des arêtes et des facettes était suffisamment prohibitif pour obliger le programmeur à le faire en dehors de l'application et à stocker les résultats dans un fichier. L'absence de types pouvant être définis par le programmeur obligeait ce dernier à stocker la description du problème dans un tableau de réels multidimensionnel, ce qui engendrait un gaspillage important de la mémoire. Enfin, un inducteur ou un aimant étant construit à partir de la juxtaposition de plusieurs milieux, un milieu correspondant à un volume défini dans le maillage, cette juxtaposition était construite sous la forme d'un produit de nombres premiers (un nombre premier étant attribué à chaque milieu), ce qui empêchait de traiter les problèmes contenant beaucoup de milieux.

Notre version Fortran 90 comporte environ 10000 lignes de code et fait un usage intensif des modules, des types définis par le programmeur ainsi que des opérations vectorielles. La figure 1.17 présente sa structure générale.

Il s'agit d'une structure complètement modulaire dans laquelle toutes les données globales de l'application sont regroupées dans un module visible depuis tous les autres, ce qui permet d'éviter les passages de paramètres propres aux versions précédentes. Cette nouvelle version ne tolère pas les déclarations implicites (`implicit none`). Un module est associé à chaque étape du code. Les formulations sont également représentées sous forme de modules autonomes, ce qui nous permettra par la suite d'ajouter les formulations magnétodynamiques sans changer la structure générale. Notons que les opérations de numérotation des inconnues, de construction et de résolution du système d'équations sont également regroupées dans un module. La numérotation des inconnues est faite en parcourant les éléments dans l'ordre donné par le mailleur. Nous verrons par la suite que ce détail a son importance pour le calcul et la stabilité de la matrice de préconditionnement.

Par rapport aux versions précédentes, nous avons ajouté la possibilité de mélanger des tétraèdres à quatre nœuds, des prismes à six nœuds et des cubes à huit nœuds au sein d'un même maillage, ce qui permet d'offrir à l'utilisateur une plus grande lattitude dans le maillage de ses structures. L'utilisation d'un nouvel algorithme, beaucoup plus efficace que le précédent, a permis de réintégrer le calcul des arêtes et des facettes au sein de l'application. La phase de discrétisation des termes sources relatifs aux inducteurs a elle aussi vu ses algorithmes d'origine remplacés par de nouveaux algorithmes beaucoup plus optimisés. Le

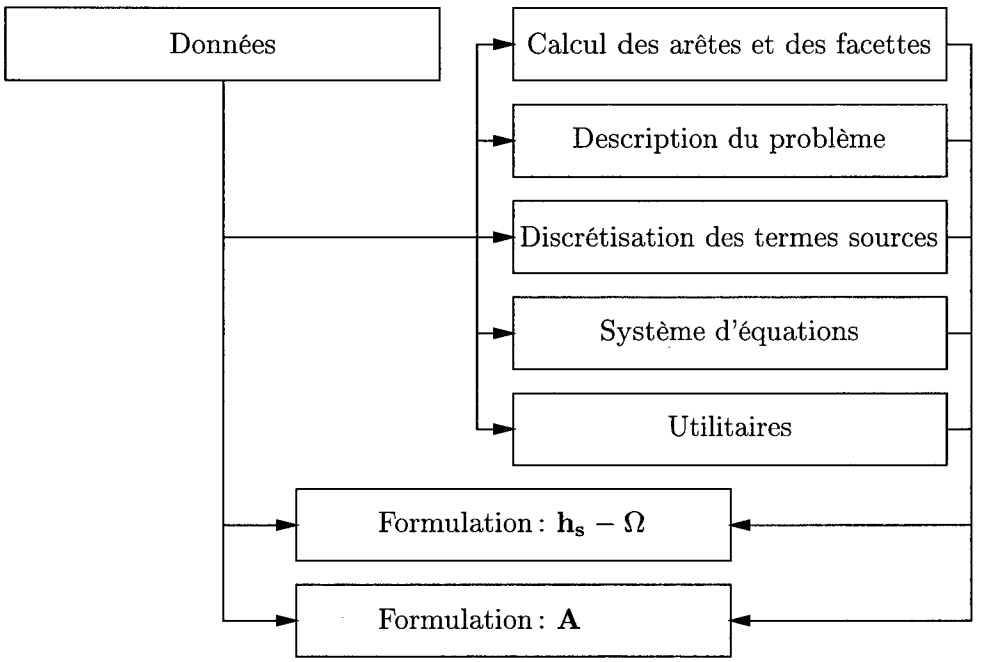

Fig. 1.17 – Structure modulaire du code

système de nombre premiers propre aux versions précédentes a également été remplacé par un autre système permettant de traiter n'importe quel dispostif.

Grâce à toutes ces modifications et nouveaux apports, nous avons constaté un accroissement très significatif des performances de cette nouvelle version par rapport aux versions précédentes. Il nous a ainsi été possible de traiter notre premier dispositif de classe véritablement industrielle : une tête de bobine pour turbo-alternateur. Notons que dans cette version, tous les tableaux manipulés sont des tableaux unidimensionnels qui contiennent des données dont le type est un type de base du langage ou un type défini par le programmeur. Comme il n'est pas possible de prévoir la taille de certains tableaux avant un instant bien précis dans l'exécution, nous n'utilisons pas l'allocation dynamique et tous les tableaux sont surdimensionnés.

1.4.2 Version HPF

En étudiant la nouvelle version Fortran 90, nous constatons que les algorithmes utilisés pour calculer les arêtes et les facettes du maillage, ainsi que ceux utilisés pour discrétiser les termes sources relatifs aux inducteurs, se prêtent mal à la parallélisation. Dans le cas du calcul des arêtes et des facettes, la difficulté est liée au fait qu'une arête et une facette peuvent être partagées par plusieurs éléments, ce qui rend délicate toute tentative de répartition des tableaux de nœuds, d'arêtes, de facettes et d'éléments de l'application. Dans le cas de la discrétisation des termes sources relatifs aux inducteurs, la difficulté est liée aux très fortes dépendances entre les calculs, dépendances inhérentes aux techniques de construction d'arbres employées. De même que précédemment, ces dépen-

dances rendent hasardeuse toute tentative de répartition des tableaux d'arêtes, de facettes et d'éléments de l'application. Dès lors, l'impossibilité de répartir les données nous contraint à les répliquer sur l'ensemble des processeurs.

La formulation en potentiel vecteur magnétique est la formulation magnétostatique la plus génératrice en terme de nombre d'inconnues. Lorsque nous utilisons cette formulation et que nous additionnons les temps de calcul relatifs au calcul des arêtes et des facettes du maillage, à la discrétisation des termes sources pour les inducteurs, à la numérotation des inconnues et à la construction de la structure CSR, nous constatons que la somme obtenue n'excède pas 10 à 15% du temps d'exécution global de l'application dans le cas linéaire. Cette constatation nous amène à penser que l'effort principal doit porter sur la parallélisation de l'assemblage et de la résolution du système d'équations.

Les caractéristiques de cette version HPF sont donc une réplication de toutes les données sur l'ensemble des processeurs et une parallélisation de l'assemblage et de la résolution du système d'équations, les autres parties du code demeurant séquentielles. Les tableaux de nœuds et d'éléments étant répliqués sur l'ensemble des processeurs, toute lecture d'un nœud ou d'un élément dans le fichier contenant le maillage provoque systématiquement une diffusion depuis le processeur chargé d'effectuer les opérations d'entrées-sorties vers les autres processeurs. Pour éviter ces diffusions multiples, nous utilisons l'extension du langage `extrinsic(hpf_serial)`, que nous avons présentée dans notre description de HPF, pour lire le fichier contenant le maillage.

Pour développer cette version HPF du code, nous avons utilisé la version 7.0 d'ADAPTOR. Le nombre de lignes de directives ou de code ajoutées à la version Fortran 90 n'excède pas 100, soit 1% du nombre de lignes original.

Assemblage

Dès que la structure CSR du système d'équations est construite, l'assemblage peut commencer. Il consiste à parcourir le tableau contenant tous les éléments du maillage, à calculer la contribution de chaque élément et à l'ajouter dans le système d'équations en cours de construction. La figure 1.18 présente la parallélisation de cet algorithme dans le cas simplifié d'une matrice non-symétrique et d'un problème sans conditions aux limites.

Dans cet exemple, nous devons calculer les tableaux A et B qui contiennent respectivement la forme compressée de la matrice et son membre droit. Pour cela, nous commençons par déclarer les tableaux elt_A et elt_B qui vont accueillir la contribution de chaque élément. La constante N représente le nombre maximum d'inconnues par élément. Le tableau $elt\_inconnues$ contient les numéros d'inconnues attribués à l'élément lors de la phase de numérotation. Le tableau contenant les éléments du maillage étant répliqué sur l'ensemble des processeurs, nous procédons comme dans l'exemple de la figure 1.9 pour répartir les itérations de la boucle. Pour cela, nous déclarons un template 1D dont la taille est donnée par le nombre d'éléments du maillage, puis nous le distribuons par blocs. L'utilisation de la directive `independent` et de la clause `on home` nous permet alors de répartir les itérations de la boucle entre les processeurs. La routine `calculer_contribution` est une routine `pure` qui accède

```
      subroutine assemblage_hpf(nb_elements)
        integer, intent(in) :: nb_elements

        real(kind = 0d0), dimension(1:N, 1:N) :: elt_A
        real(kind = 0d0), dimension(1:N) :: elt_B
        integer, dimension(1:N) :: elt_inconnues
        integer :: elt_N, i, j, k, lin, col, pos
!hpf$ template t(1:nb_elements)
!hpf$ distribute(block) :: t

        A = 0.0d0
        B = 0.0d0
!hpf$ independent, on home(t(i)), reduction(A, B) &
!hpf$ new(elt_A, elt_B, elt_inconnues, elt_N, j, k, lin, col)
        do i = 1, nb_elements
          call calculer_contribution(i, elt_A, elt_B, elt_inconnues, elt_N)
          do j = 1, elt_N
            lin = elt_inconnues(j)
            B(lin) = B(lin) + elt_B(j)
            do k = 1, elt_N
              col = elt_inconnues(k)
              call trouver_position_CSR(lin, col, pos)
              A(pos) = A(pos) + elt_A(j, k)
            end do
          end do
        end do
      end subroutine assemblage_hpf
```

Fig. 1.18 – Parallélisation de l'assemblage, cas simplifié

à la ième position du tableau contenant les éléments du maillage, qui calcule la contribution de l'élément correspondant et qui collecte sa liste d'inconnues et le nombre de ces dernières (le maillage pouvant comporter différents types d'éléments). Nous assemblons ensuite le membre droit B puis la matrice A. La routine `trouver_position_CSR` est elle aussi une routine `pure` qui calcule la position de l'entrée $M(lin, col)$ de la matrice dans la structure CSR. Les tableaux contenant respectivement la matrice compressée A et le membre droit B étant également répliqués sur l'ensemble des processeurs, comme pour l'exemple de la figure 1.9, nous appliquons la directive `reduction` à ces deux tableaux pour obtenir le système final. L'ordre d'exécution de la boucle ne change pas le résultat final, aux erreurs d'arrondis près.

Dans la réalité, nous ne stockons les entrées $M(lin, col)$ de la partie triangulaire inférieure de M ($col \leq lin$) que s'il n'existe pas de condition à la limite homogène associée aux inconnues `elt_inconnues(j)` et `elt_inconnues(k)`. La numérotation des inconnues commençant à partir de un, toute inconnue avec une valeur nulle signale une telle condition.

Résolution

La résolution par la méthode du gradient conjugué préconditionné, est dominée par trois types d'opérations :

- des produits scalaires ;
- des produits matrice-vecteur $b = Mx$ où M désigne la matrice du système d'équations ;
- des résolutions triangulaires $Nx = b$ où N désigne la matrice de préconditionnement.

Pour paralléliser les produits scalaires, nous alignons tous les vecteurs utilisés par le gradient conjugué et nous les répartissons par blocs. Les opérateurs vectoriels tels que `dot_product` étant déjà parallélisés dans ADAPTOR, nous n'avons pas à écrire explicitement le code du produit scalaire.

Pour paralléliser les produits matrice-vecteur, nous devons répartir la matrice creuse M par blocs de lignes en lui appliquant une directive de répartition par blocs généralisée. Ainsi, sa ième ligne et les éléments de vecteur $x(i)$ et $b(i)$ appartiendront au même processeur. Nous avons remarqué que dans la majorité des exemples traités, la différence relative au nombre d'éléments de deux lignes de la matrice M était généralement petite. De fait, une répartition par blocs de lignes permet d'obtenir un bon équilibrage de charge. La figure 1.19 présente cette répartition sur trois processeurs.

Pour répartir la matrice M par blocs de lignes, nous devons connaître le nombre de processeurs utilisés pour exécuter le code, afin de calculer la taille de chaque bloc. Le premier problème qui se pose est lié au fait que les tableaux A et jA sont statiques, surdimensionnés et répliqués sur l'ensemble des processeurs. Le surdimensionnement est lié au fait qu'en début d'exécution, seuls les nombres de nœuds et d'éléments sont connus. Les nombres d'arêtes et de facettes ne peuvent, quant à eux, être connus qu'a la fin de la phase de calcul de ces dernières. De plus, l'utilisation de conditions aux limites ne permet pas de connaître le nombre d'entrées de la matrice M avant la fin de son assemblage. Dès lors,

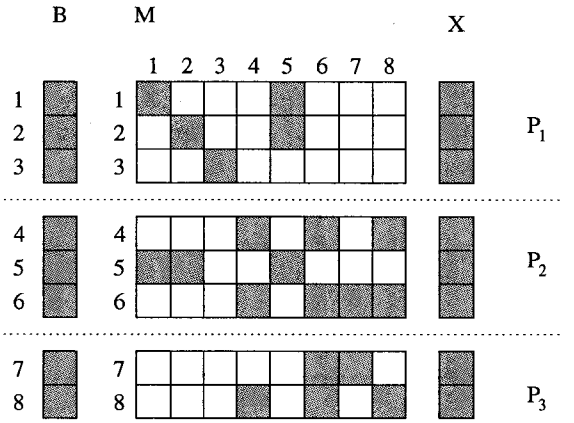

Fig. 1.19 – Répartition sur 3 processeurs

les tableaux correspondants doivent être surdimensionnés. Ces derniers n'étant pas dynamiques et n'ayant pas fait l'objet d'une directive de répartition initiale, il n'est pas possible de leur appliquer la directive de re-répartition dynamique `redistribute`. Nous devons alors envisager de recourir à de nouveaux tableaux A_d et jA_d, dynamiques cette fois, analogues à ceux utilisés dans l'exemple de la figure 1.11 et qui feront l'objet d'une directive de répartition initiale par blocs, tout en sachant qu'une fois alloués et initialisés, ils feront l'objet de la directive de re-répartition dynamique `redistribute`. Le second problème qui se pose est lié au surdimensionnement des vecteurs utilisés par le gradient conjugué. Si ces vecteurs étaient alloués exactement à la taille donnée par le nombre d'inconnues du problème, alors une directive initiale de répartition par blocs permettrait d'obtenir la répartition des éléments de vecteurs que nous recherchons, le compilateur se servant implicitement du nombre de processeurs utilisés pour calculer la taille de chaque bloc. Dans notre cas, le surdimensionnement fausserait ce calcul. En effet, supposons que dans l'exemple de la figure 1.19, les tableaux b et x soient surdimensionnés pour accueillir non pas huit mais douze éléments, la répartition par blocs conduirait alors à attribuer quatre éléments consécutifs à chaque processeur, répartition qui serait incompatible avec celle de la matrice M. Là encore, nous devons envisager de recourir à de nouveaux tableaux dynamiques. Cependant, contrairement aux tableaux A_d et jA_d, ces nouveaux tableaux ne feront l'objet que d'une directive initiale de répartition par blocs.

La déclaration de nouveaux tableaux dynamiques, leur allocation à la taille voulue et leur initialisation, ne peuvent se faire par le simple ajout de directives HPF, mais par l'ajout de nouvelles lignes de code dans l'application d'origine. Pour connaître le nombre de processeurs utilisés, nous faisons appel à la fonction `number_of_processors` du Fortran 90. Comme auparavant, cette version HPF pourra être compilée et exécutée sur une machine mono-processeur. Les différences avec la version d'origine seront une plus grande consommation de mémoire et certains calculs inutiles.

Une fois la répartition de la matrice M effectuée, l'obligation pour chaque processeur de se procurer les éléments de vecteurs $x(j)$ non-locaux qui correspondent à ses entrées $M(i,j)$ non-nulles de la matrice, est à l'origine des communications inter-processeurs. La figure 1.20 présente l'utilisation de Halos pour le produit matrice-vecteur ($b = Mx$) sur trois processeurs. Le tableau d'indices utilisé est jA_d. De la mémoire "shadow" est allouée pour chaque nouveau vecteur utilisé par le gradient conjugué. Ces vecteurs étant répartis de façon identique, les informations collectées durant la phase d'inspection pour l'un de ces vecteurs, sont valables pour tous les autres.

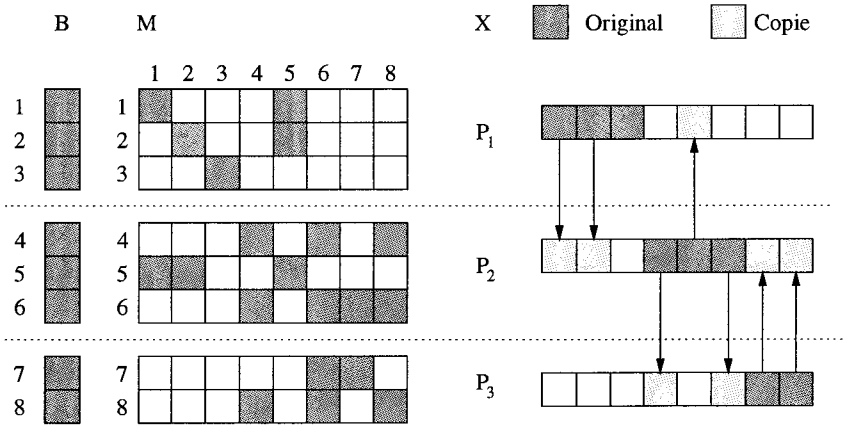

Fig. 1.20 – Halos sur 3 processeurs

La figure 1.21 présente la parallélisation HPF-Halos du produit matrice-vecteur. Nous supposons dans cette figure que la phase de création des mémoires "shadow" ainsi que la phase d'inspection ont déjà été réalisées. Le tableau iA n'ayant pas fait l'objet d'une directive de répartition, il est répliqué sur l'ensemble des processeurs.

Le produit matrice-vecteur est effectué en deux phases. La première est relative à la partie triangulaire inférieure de la matrice. Nous utilisons la directive `independent` et la clause `on home` sur la donnée $b\_d(i)$ pour répartir les itérations de la boucle entre les processeurs. Les tableaux b_d, x_d, A_d et jA_d ayant des tailles différentes, il est impossible de les aligner. L'utilisation de la clause `resident` empêche alors le compilateur de supposer l'existence de communications inter-processeurs pour accèder aux éléments des tableaux A_d et jA_d. Une mise à jour effectuée par Halos en amont de cette boucle, permet de recopier la valeurs des éléments originaux du tableau x_d dans leurs différentes copies. La seconde phase est relative à la partie triangulaire supérieure de la matrice M. Cette dernière n'étant pas stockée explicitement, nous devons utiliser sa transposée. De fait, contrairement à la boucle précédente, la clause `on home` concerne ici la donnée $x(i)$. L'utilisation de la transposée de M nécessite d'initialiser à zéro les différentes copies des éléments originaux du tableau b_d en amont de la boucle (`halo_init`), puis de leur appliquer une réduction en aval afin d'obtenir le vecteur final. Ces deux opérations sont respectivement effectuées via Halos.

```
      subroutine matrice_vecteur_hpf
        integer :: i, j

        call halo_update(id, x_d)

!hpf$ independent, on home(b_d(i)), resident(A_d, jA_d, x_d), new(j)
        do i = 1, nb_inconnues
          b_d(i) = 0.0d0
          do j = iA(i), iA(i + 1) - 1
            b_d(i) = b_d(i) + A_d(j) * x_d(jA_d(j))
          end do
        end do

        call halo_init(id, b_d, 0.0d0)

!hpf$ independent, on home(x_d(i)), resident(A_d, jA_d, b_d), new(j)
        do i = 1, nb_inconnues
          do j = iA(i), iA(i + 1) - 2
            b_d(jA_d(j)) = b_d(jA_d(j)) + A_d(j) * x_d(i)
          end do
        end do

        call halo_reduce_sum(id, b_d)
      end subroutine matrice_vecteur_hpf
```

Fig. 1.21 – Parallélisation HPF-Halos du produit matrice-vecteur

Les résolutions triangulaires ne peuvent être parallélisées en l'état en raison des dépendances entre les calculs. Ces dépendances peuvent cependant être supprimées si nous parvenons à éliminer certaines entrées pour obtenir une matrice diagonale par blocs, ainsi que le montre la figure 1.22. Il s'agit alors d'un préconditionneur par blocs analogue au préconditionneur par blocs de Jacobi, à l'exception près que pour calculer les entrées de chaque bloc, nous utilisons des entrées qui sont extérieures à ces blocs.

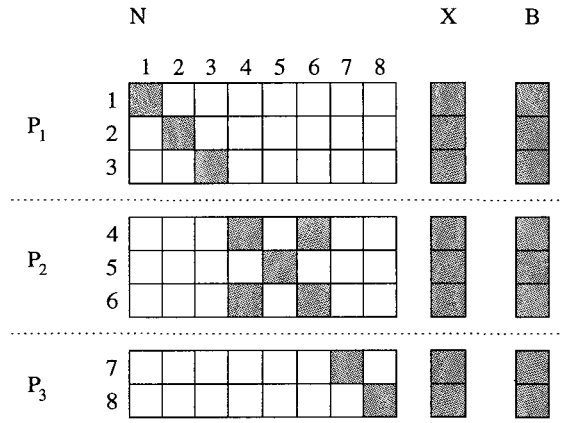

Fig. 1.22 – Matrice diagonale par blocs sur 3 processeurs

Comme pour la forme compressée A de la matrice M, un nouveau tableau dynamique C_d est associé à la forme compressée C de la matrice de préconditionnement N. Ce tableau est aligné sur le tableau A_d et fait donc l'objet de la même directive de re-répartition dynamique par blocs généralisée. Pour obtenir une matrice diagonale par blocs depuis la matrice N, cette dernière étant répliquée sur l'ensemble des processeurs, nous appliquons la technique du probing [26]. L'idée de cette technique est de supprimer toutes les entrées d'une matrice situées au delà d'une certaine distance de la diagonale, en la multipliant par une autre matrice contenant des vecteurs dits de probing. Dans notre cas, il s'agit de supprimer toutes les entrées de la matrice de préconditionnement qui nécessitent des éléments de vecteurs non-locaux. Cette technique peut être mise en oeuvre par l'algorithme de la figure 1.23.

Cet algorithme tire partie du fait que la matrice de préconditionnement N est répliquée sur l'ensemble des processeurs et qu'elle possède la même structure que la matrice M. Dès lors, il est possible d'utiliser un vecteur réparti par blocs et sa mémoire "shadow" comme vecteur de probing. La figure 1.24 permet d'illustrer ce principe.

Dans cette figure, la matrice N qui possède la même structure que la matrice M, est répliquée sur l'ensemble des processeurs. Par le jeu des mémoires "shadow", nous pouvons créer un vecteur de probing différent pour chaque processeur. En initialisant à la valeur un les éléments locaux de ce vecteur, nous conservons les entrées $N(i, j)$ ne nécessitant que des éléments de vecteurs locaux. En initialisant à la valeur zéro la mémoire "shadow" de ce vecteur, nous supprimons les entrées $N(i, j)$ nécessitant des éléments de vecteurs non-locaux.

```
      subroutine probing_hpf
        intger :: i, j

        x_d = 1.0d0
        call halo_init(id, x_d, 0.0d0)

!hpf$ independent, on home(x_d(i)), resident(C_d, jA_d), new(j)
        do i = 1, nb_inconnues
          do j = iA(i), iA(i + 1) - 1
            C_d(j) = C(j) * x_d(jA_d(j))
          end do
        end do
      end subroutine probing_hpf
```

Fig. 1.23 – Technique du probing

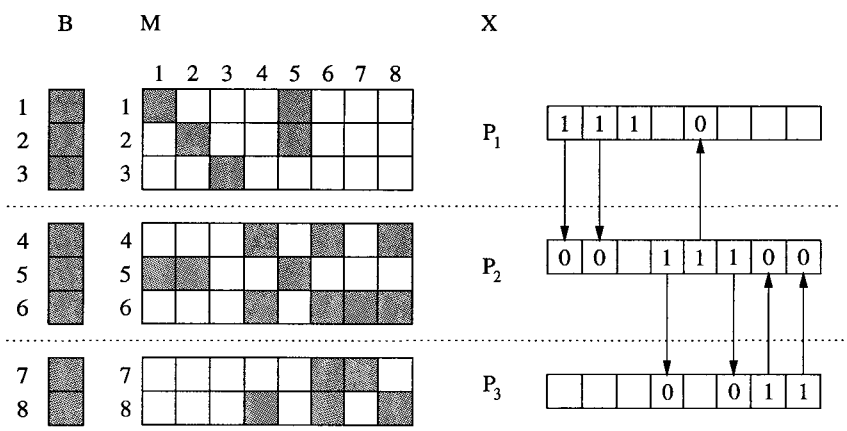

Fig. 1.24 – Probing sur 3 processeurs

A l'issue de la multiplication de la matrice N par nos vecteurs de probing, nous obtenons la matrice diagonale par blocs de la figure 1.22.

1.4.3 Résultats

Pour tester la version HPF du code, nous avons utilisé des dispositifs réels. Nous présentons ici les résultats obtenus pour deux de ces dispositifs : une bobine à noyau de fer et une machine synchrone à aimants permanents, dans le cas linéaire et en utilisant une formulation en potentiel vecteur magnétique. Les machines parallèles utilisées sont celles du GMD. Il s'agit d'une SP2 d'IBM (machine à mémoire répartie) et une Origin 2000 de Silicon Graphics (machine à base de SMP CC-NUMA) à quatre processeurs. Dans les deux cas, la nouvelle version du code générée par ADAPTOR est basée sur le modèle à parallélisme de tâches et la bibliothèque de communication MPI.

Tous les résultats que nous présentons ici ont été vérifiés graphiquement grâce aux fichiers de vecteurs générés par l'application et numériquement par le calcul de l'énergie électromagnétique du dispositif.

Bobine à noyau de fer

La figure 1.25 présente le dispositif considéré. Il s'agit d'une bobine à noyau de fer. Pour des raisons de symétrie, le domaine d'étude est ramené à une moitié de cette bobine. Des conditions de continuité sont assurées sur le plan de symétrie.

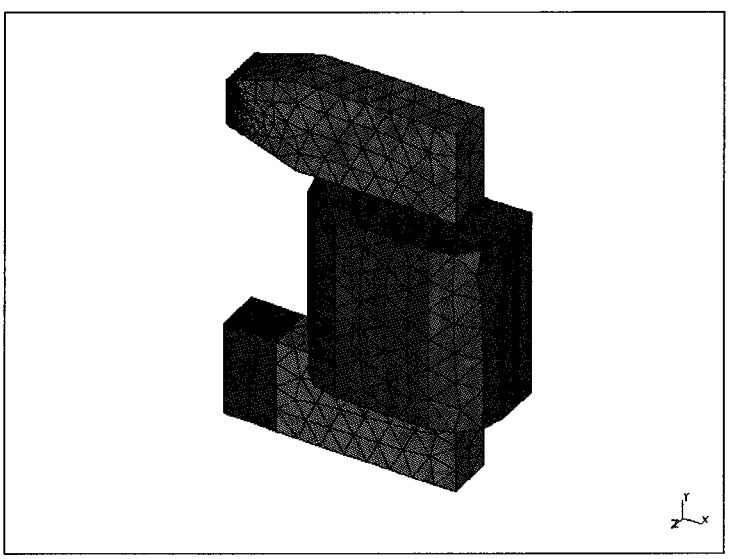

Fig. 1.25 – Bobine à noyau de fer

La table 1.1 présente les caractéristiques du problème à traiter.

Les tables 1.2 et 1.3 présentent les résultats respectivement obtenus sur l'Origin 2000 et la SP2. Y sont donnés, en fonction du nombre de processeurs utilisés, le temps d'exécution de la boucle d'assemblage, le temps d'exécution de

Tab. 1.1 – Caractéristiques du problème

nœuds	8059
arêtes	51356
facettes	84620
éléments	41322
inconnues	49480
entrées non-nulles	412255

la résolution, le nombre d'itérations du gradient conjugué et le temps passé dans chaque itération. Les tables 1.4 et 1.5 présentent, quant à elles, les accélérations obtenues pour la phase d'assemblage, pour la phase de résolution et pour chaque itération du gradient conjugué sur ces mêmes machines.

Tab. 1.2 – Résultats sur l'Origin 2000

	NP = 1	NP = 2	NP = 3	NP = 4
assemblage	12,87 s	6,54 s	4,63 s	3,61 s
résolution	30,47 s	18,89 s	9,74 s	8,80 s
itérations	225	260	213	237
temps/itération	135 ms	71 ms	45 ms	37 ms

Tab. 1.3 – Résultats sur la SP2

	NP = 1	NP = 2	NP = 4	NP = 8	NP = 16
assemblage	32,70 s	16,77 s	8,98 s	5,27 s	3,47 s
résolution	48,45 s	36,99 s	23,51 s	14,45 s	10,59 s
itérations	224	247	257	247	258
temps/itération	216 ms	149 ms	91 ms	58 ms	41 ms

La comparaison des résultats obtenus sur ces deux machines met en évidence plusieurs phénomènes :

1. à nombre de processeurs égal, le nombre d'itérations du gradient conjugué diffère selon les machines ;
2. le poids de la directive `reduction` utilisée dans la boucle d'assemblage diffère selon les machines ;
3. les accélérations enregistrées pour la phase d'assemblage restent assez proches du nombre de processeurs quand ce dernier est petit. Elles s'en éloignent assez régulièrement lorsque ce nombre augmente;
4. les accélérations obtenues pour la phase de résolution sont acceptables pour un petit nombre de processeurs. Elles se dégradent fortement quand ce nombre augmente.

Pour expliquer ces phénomènes, nous pouvons avancer les explications suivantes :

1. la différence relative au nombre d'itérations du gradient conjugué sur ces deux machines, s'explique par l'agressivité des optimisations (niveau `O3`) mises en oeuvre par les compilateurs Fortran 90 natifs de ces machines.

Tab. 1.4 – Accélérations sur l'Origin 2000

	NP = 1	NP = 2	NP = 3	NP = 4
assemblage	1,00	1,96	2,77	3,56
résolution	1,00	1,61	3,12	3,46
itération	1,00	1,9	3,00	3,64

Tab. 1.5 – Accélérations sur la SP2

	NP = 1	NP = 2	NP = 4	NP = 8	NP = 16
assemblage	1,00	1,94	3,64	6,20	9,42
résolution	1,00	1,30	2,06	3,35	4,57
itération	1,00	1,44	2,37	3,72	5,26

Si le niveau `O2` garantit le résultat à la décimale près, il n'en n'est pas de même du niveau `O3` ;

2. à la différence de la SP2 qui est une machine à mémoire répartie classique, l'Origin 2000 est une machine à base de SMP CC-NUMA. Lorsque deux, trois ou quatre tâches sont lancées sur la SP2, ces tâches doivent obligatoirement utiliser le réseau d'interconnexion pour communiquer entre elles. Lorsque ces mêmes tâches sont lancées sur l'Origin 2000, elles le sont sur le même nœud de calcul. Les communications entre ces tâches passent donc par la mémoire partagée du nœud et non par le réseau d'interconnexion entre les différents nœuds de calcul de la machine. Elles sont alors beaucoup plus efficaces que sur une machine à mémoire répartie classique telle que la SP2 ;
3. nous constatons que jusqu'à quatre processeurs, les accélérations obtenues sur les deux machines sont relativement proches de ce nombre, l'avantage revenant à l'Origin 2000 en raison de communications plus efficaces. Nous constatons également qu'au-delà de quatre processeurs utilisés sur la SP2, le poids de la réduction devient très important. Cet effet montre qu'au-delà d'un certain nombre de processeurs utilisés, la qualité du réseau de communication ne peut plus grand chose face à l'accroissement des volumes de données échangées ;
4. la numérotation des inconnues du problème est effectuée en parcourant le tableau contenant tous les éléments du maillage. Le numéro attribué à un nœud ou à une arête de l'élément, correspond à une position dans les vecteurs utilisés par le gradient conjugué. Ce numéro est utilisé pour construire la structure CSR du système d'équations, et notamment le tableau des indices de colonnes jA qui est à l'origine des communications inter-processeurs de la phase de résolution. La numérotation des inconnues n'étant pas couplée à la répartition des données, le nombre de communications inter-processeurs est imprévisible. Dès lors, plus le nombre de processeurs augmente et plus le nombre d'éléments de vecteurs non-locaux s'accroît sur chaque processeur, entraînant une augmentation générale des volumes de données échangées entre les processeurs. Cette augmentation, non-contrôlable actuellement, explique la détérioration des accélérations

observée sur la SP2.

Machine synchrone à aimants permanents

La figure 1.26 présente le dispositif considéré. Dans ce cas, il s'agit d'une machine synchrone à aimants permanents. Pour des raisons de symétrie, le domaine d'étude est ramené à un huitième de cette machine. Des conditions de continuité sont assurées sur les plans de symétrie.

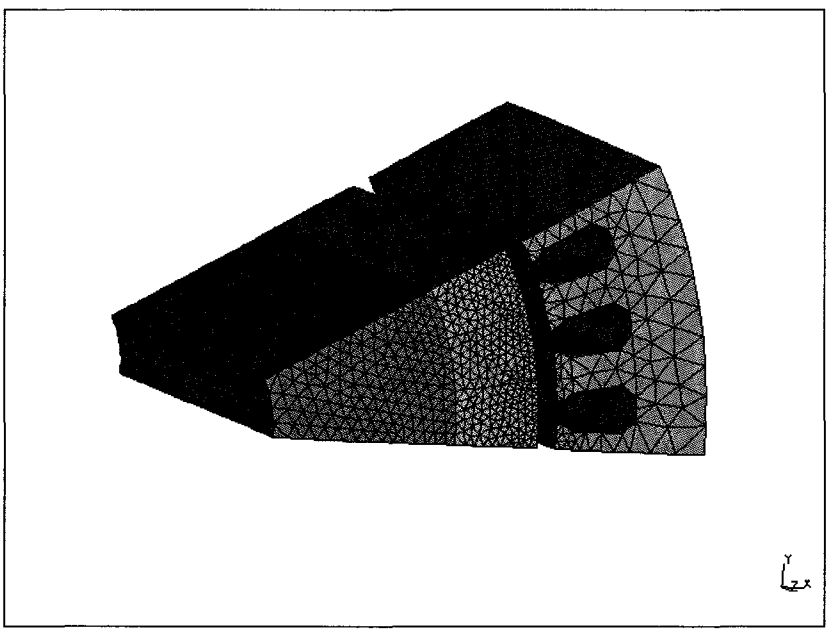

Fig. 1.26 – machine synchrone à aimants permanents

La table 1.6 présente les caractéristiques du problème à traiter.

Tab. 1.6 – Caractéristiques du problème

nœuds	25730
arêtes	169942
facettes	284669
éléments	140456
inconnues	162939
entrées non-nulles	1382596

Ce dispositif est beaucoup plus important que le précédent. Les résultats obtenus pour la bobine à noyau de fer sur la SP2 d'IBM permettent de dire que les résultats obtenus sur cette machine pour notre nouvel exemple seront de toutes façons décevants. De fait, nous ne l'avons exécuté que sur l'Origin 2000. Les tables 1.7 et 1.8 présentent respectivement les résultats et les accélérations obtenues sur cette machine.

Les résultats présentés dans ces tables peuvent paraître surprenants. Pour comprendre la raison de la chute brutale du nombre d'itérations du gradient

Tab. 1.7 – Résultats sur l'Origin 2000

	NP = 1	NP = 2	NP = 3	NP = 4
assemblage	43,77 s	22,71 s	16,03 s	12,95 s
résolution	625,06 s	160,84 s	116,63 s	93,25 s
itérations	1174	464	509	505
temps/itération	532 ms	347 ms	229 ms	165 ms

Tab. 1.8 – accélérations sur l'Origin 2000

	NP = 1	NP = 2	NP = 3	NP = 4
assemblage	1,00	1,92	2,73	3,37
résolution	1,00	3,88	5,35	6,70
par itération	1,00	1,53	2,32	3,22

conjugué, il faut s'intéresser une fois de plus à la numérotation des inconnues, cette numérotation jouant un rôle primordial dans le conditionnement de la matrice du système d'équations. Dans notre cas, le mailleur énumère les éléments par volumes, chaque volume correspondant à un matériau (air, cuivre, fer, etc.) et donc à une perméabilité magnétique. Afin de ne pas introduire de multiples sauts de coefficients à l'intérieur de la matrice, la numérotation des inconnues s'effectue en parcourant le tableau contenant tous les éléments du maillage. De fait, à chaque volume du maillage correspond un bloc homogène de la matrice. Le conditionnement de cette dernière est d'autant meilleur que les perméabilités magnétiques des différents blocs sont proches. Il se dégrade d'autant plus vite que ces perméabilités magnétiques sont disparates. Les préconditionneurs par blocs issus des méthodes de Schwarz permettent de résoudre ce problème. Leur principe est de préconditionner séparément chaque bloc de la matrice (moyennant une numérotation adéquate des inconnues) afin de circonscrire l'instabilité liée aux sauts de coefficients à l'intérieur de ces blocs et de ne pas la communiquer au reste du préconditionneur. Dans notre cas, le préconditionneur que nous utilisons est analogue au préconditionneur de Jacobi par blocs. Pour vérifier que la chute brutale du nombre d'itérations du gradient conjugué était bien le fait de ce préconditionneur, nous avons utilisé la seconde version HPF du code, celle-ci étant basée sur une méthode de Schwarz. Nous avons effectivement constaté qu'en utilisant un préconditionneur de Jacobi par blocs, le nombre de ces itérations était bel et bien divisé par deux. Certains mailleurs sont munis de modules de résolution basés sur des méthodes directes (méthode de Gauss, de Cholesky, etc.) et proposent des options permettant de diminuer la largeur de bande des matrices par une re-numérotation adéquate des nœuds. Dans notre cas, ces optimisations ne nous sont d'aucun secours puisque nous utilisons une méthode de résolution itérative et que nos inconnues peuvent également concerner les arêtes du maillage.

Les accélérations obtenues pour la phase d'assemblage sont identiques à celles obtenues pour la bobine à noyau de fer. Par contre, les accélérations que nous obtenons pour la phase de résolution sont surlinéaires. Elles sont dûes à la chute brutale du nombre d'itérations du gradient conjugué. Pour deux pro-

cesseurs et plus, nous constatons que le temps passé dans chaque itération du gradient conjugué suit la même évolution que pour la bobine à noyau de fer.

1.5 Conclusion

La section 1.2 à présenté le code que nous devions paralléliser. Il s'agissait d'un code éléments finis 3D, écrit en Fortran 77 et développé par le Laboratoire d'Electrotechnique et d'Electronique de Puissance de Lille (L2EP).

Les choix que nous sommes amenés à faire en programmation parallèle, sont guidés par plusieurs critères et forment généralement le meilleur compromis possible au vu de ces derniers. Ces critères peuvent être les caractéristiques de l'application, les critères classiques du génie logiciel et le public concerné. L'application que nous devions paralléliser était un code de recherche irrégulier en constante évolution. Les critères que nous avons alors définis étaient la facilité de développement et de maintenance alliée à une certaine efficacité. Nos choix ont été arrêtés sur la facilité de développement et de maintenance inhérente au modèle à parallélisme de données incarné par le langage standard High Performance Fortran (HPF) [38] (les compilateurs HPF étant libres dans le choix du modèle d'exécution), l'efficacité et la simplicité d'utilisation de Halos [3], une bibliothèque de communication pour les applications irrégulières basées sur le parallélisme de données (ces bibliothèques étant plus souvent l'apanage des applications basées sur le parallélisme de tâches), ainsi que le système de compilation HPF ADAPTOR [1]. Dans la section 1.3, nous avons détaillé ces critères et ces choix en expliquant pourquoi ces derniers formaient à nos yeux le meilleur compromis possible.

La section 1.4 a présenté une approche parallèle légère de la partie magnétostatique de l'application, ainsi que les résultats obtenus pour cette dernière. Par approche parallèle légère, nous entendons une parallélisation ne remettant pas en cause la structure et les algorithmes fondamentaux de l'application, la seule opération autorisée étant l'ajout, en quantité limitée, de directives HPF ou de nouvelles lignes de code. Bien que nous sachions que les meilleures versions parallèles des codes éléments finis sont basées sur les méthodes de Schwarz, une telle approche permet d'une part de rendre abordable la programmation parallèle à une communauté de non-informaticiens. Elle permet d'autre part de valider le couplage entre le modèle à parallélisme de donnée incarné par HPF et les bibliothèques de communication incarnées par Halos, pour une application de grande envergure. Enfin, elle permet de prévoir les changements de structures et d'algorithmes qui seront nécessaires à l'introduction des méthodes de Schwarz.

Les résultats obtenus pour cette approche montrent que ses objectifs ont été atteints. D'une part, le nombre de directives ou de nouvelles lignes de code ajoutées n'excède pas 1% du nombre de lignes de la version Fortran 90 que nous avons ré-écrite. De fait, cette version HPF est pratiquement identique à son homologue Fortran 90, ce qui permet à des non-informaticiens d'en assurer le développement et la maintenance. D'autre part, les performances obtenues pour cette version sans localité des données, ont mis en évidence des accélérations tout à fait acceptables pour un petit nombre de processeurs, ce qui valide le

couplage HPF-Halos. Ces résultats indiquent également un équilibrage de charge relativement bon et implicite. Il est la conséquence de l'utilisation d'un template réparti par blocs dans la phase d'assemblage du système d'équations. Il est également la conséquence d'une répartition par blocs généralisées des matrices dans la phase de résolution, ces dernières étant suffisamment creuses pour que la différence relative au nombre d'éléments de deux de leurs lignes soit petite.

La première faiblesse de cette version est une consommation de mémoire importante du fait de la réplication des données et de l'allocation dynamique de nouveaux tableaux. Cette consommation est suffisamment importante pour nous empêcher de traiter de grands problèmes. Sa seconde faiblesse est liée à l'absence de localité des données, ce qui engendre des volumes de communications inter-processeurs importants quand le nombre de ces derniers augmente. Cette localité ne peut être contrôlée que par l'intermédiaire de la phase de numérotation des inconnues, mais un tel contrôle réclame une certaine connaissance du maillage que nous n'avons pas dans cette version. Enfin, Halos est directement présente dans le code même de l'application, ce qui rend cette dernière complétement dépendante. Cette dépendance fait que l'utilisation d'une autre bibliothèque ou d'une autre méthode permettant d'échanger les informations entre les processeurs, par exemple une réplication des données concernées sur l'ensemble des processeurs, exige une modification de plusieurs modules de l'application.

Chapitre 2

Approche parallèle lourde

2.1 Introduction

L'approche parallèle légère présentée dans le chapitre 1, consiste à paralléliser une application en ne changeant rien de sa structure et de ses algorithmes fondamentaux. La seule opération autorisée est l'ajout, en faibles quantités, de directives HPF et de nouvelles lignes de code. Cette approche permet de valider le couple HPF-Halos, HPF incarnant la facilité de développement et de maintenance du modèle de programmation à parallélisme de données et Halos incarnant l'efficacité des bibliothèques de passage de messages. Les résultats obtenus montrent des accélérations acceptables pour des petits nombres de processeurs, alors que le nombre de directives et de lignes ajoutées dans le code n'excède pas 1% du nombre de lignes original. Cette approche présente cependant certaines limitations :

- l'impossibilité de répartir les données pour paralléliser la phase de calcul des arêtes et des facettes du maillage ainsi que celle relative à la discrétisation des termes sources, nous impose de répliquer ces données sur l'ensemble des processeurs utilisés. D'une part, une telle démarche nous interdit de traiter des problèmes très importants. D'autre part, elle pénalise la parallélisation de la boucle d'assemblage du fait de l'utilisation de la directive `reduction` ;
- l'absence de localité des données engendre des volumes de communications importants qui sont échangés au cours de la résolution, ce qui empêche l'application de s'exécuter sur un grand nombre de processeurs ;
- l'omniprésence de Halos dans les directives HPF (utilisation de la directive `resident` pour les mémoires "shadow") et dans les lignes de code ajoutées, rend l'application complètement dépendante de cette dernière.

Pour supprimer la réplication des données et obtenir la localité de ces dernières, nous présentons une seconde approche basée sur les méthodes additives de Schwarz. Ces méthodes imposent de modifier la structure et les algorithmes fondamentaux de l'application d'origine et donc de ré-écrire une nouvelle version Fortran 90. Dès lors, une telle approche peut être qualifiée d'approche parallèle lourde, par opposition à l'approche précédente qui consistait à paralléliser un code existant. Dans cette nouvelle approche, nous devons assurer un compromis

entre simplicité de développement et de maintenance et efficacité. Pour l'obtenir, nous ré-écrivons la nouvelle version Fortran 90 en fonction de la manière dont nous voulons la paralléliser en HPF-Halos. Dans cette version, les zones de code jugées peu évolutives (lecture du maillage, calcul de ses arêtes et de ses facettes, communications, etc.) sont regroupées dans des modules "boites noires". Les zones supposées évolutives emploient, quant à elles, des algorithmes aussi similaires que possible à ceux utilisés dans l'approche précédente. Notre nouvelle version Fortran 90 étant écrite en vue de sa parallélisation en HPF-Halos, cette étape est beaucoup plus facile à réaliser que dans l'approche précédente.

Les problèmes que nous traitons étant elliptiques du second ordre, nous avons le choix entre toutes les méthodes additives de Schwarz. Nous procédons alors par élimination pour déterminer une méthode particulière qui convienne à notre application. La section 2.2 présente ce processus d'élimination et la méthode retenue : la méthode du complément de Schur.

Deux hypothèses numériques sous-tendent notre nouvelle approche. La première est relative à la discrétisation des termes sources. Elle consiste à remplacer les arbres de facettes et d'arêtes utilisés dans l'approche précédente, par des forêts d'arbres de facettes et d'arêtes, ces dernières pouvant être construites quasi-simultanément. La seconde est relative à la matrice du système d'équations et à la matrice de préconditionnement associée. Elle consiste à dire, d'une part, que la méthode du complément de Schur exige des interfaces de forme régulière entres les sous-domaines. Elle consiste à dire, d'autre part, que les dispositifs électrotechniques que nous modélisons sont rarement des dispositifs complets, ceux-ci possédant généralement des plans de symétrie. Ces deux constatations permettent alors de dire que le maillage devra être partitionné manuellement par l'utilisateur et que le nombre de sous-domaines employés (et donc de blocs), sera peu important. Dès lors, en résolvant le système d'équations global relatif à toutes les inconnues des sous-domaines comme le préconise Y. Saad [57] et non le système réduit aux seules inconnues des interfaces entre ces sous-domaines, nous pouvons ne pas utiliser le sous-espace V_0 des méthodes de Schwarz, sous-espace présenté en partie I. Ce dernier est source de communications entre les sous-domaines et les opérateurs d'interpolation sous-jacents doivent être adaptés à la dimension trois. La section 2.3 présente la nouvelle version Fortran 90 du code. Cette version, qui repose sur les deux hypothèses présentées ci-dessus, est écrite en fonction de la manière dont nous voulons la paralléliser en HPF-Halos et en fonction d'un petit nombre de sous-domaines, de l'ordre de quelques dizaines.

Les section 2.4 et 2.5 présentent respectivement la parallélisation HPF-Halos de la nouvelle version Fortran 90 et les résultats obtenus pour cette dernière. Le type de parallélisme employé est ici un parallélisme au niveau des sous-domaines.

2.2 Choix d'une méthode additive de Schwarz

Les méthodes additives de Schwarz que nous avons présentées en partie I, permettent de construire, moyennant une numérotation adéquate des inconnues, des matrices avec une structure par blocs et des préconditionneurs associés. Une

telle structure est parfaitement adaptée aux architectures parallèles. Elle permet d'obtenir la localité des données qui nous faisait défaut dans l'approche parallèle légère présentée dans le chapitre 1. Nos problèmes étant elliptiques du second ordre, nous avons le choix entre toute la panoplie des méthodes additives de Schwarz, qu'elles soient multi-niveaux ou de sous-domaines. Aussi, pour identifier une méthode particulière qui convienne à notre application, nous allons procéder par élimination.

Les maillages que nous utilisons n'étant pas uniformes, nous pouvons éliminer la catégorie des méthodes multi-niveaux. Reste alors la catégorie des méthodes de sous-domaines avec ou sans recouvrement. Pour éliminer l'une ou l'autre de ces sous-catégories, nous devons mesurer son adéquation avec la phase de calcul des arêtes et des facettes du maillage, ainsi qu'avec celle relative à la discrétisation des termes sources.

Concernant la discrétisation de la densité de courant, nous pouvons imaginer de découper le maillage d'une manière telle qu'un processeur se voit attribuer un inducteur complet du dispositif considéré. Une telle façon de faire permet de ne pas changer les algorithmes utilisés dans la version Fortran 90 du code. Le problème qui se pose alors est celui du "cube inducteur". Il s'agit d'un cas test dans lequel un cube, doté de certaines caractéristiques, est assimilé à un inducteur parcouru par une densité de courant bien précise. La solution de ce problème pouvant être calculée analytiquement, elle permet de valider les codes éléments finis 3D. Avec la solution que nous imaginons, il nous est impossible de traiter un pareil cas. Il devient alors évident qu'un même inducteur doit pouvoir être réparti sur plusieurs processeurs.

Considérons à présent l'exemple de la figure 2.1 dans lequel les éléments $e_0 = \{n_1, n_2, n_5, n_6, n_9, n_{10}, n_{13}, n_{14}\}$, $e_1 = \{n_2, n_3, n_6, n_7, n_{10}, n_{11}, n_{14}, n_{15}\}$ et $e_2 = \{n_3, n_4, n_7, n_8, n_{11}, n_{12}, n_{15}, n_{16}\}$ forment un même inducteur.

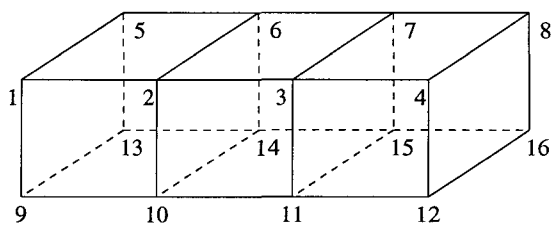

Fig. 2.1 – Inducteur

Supposons que nous utilisions une méthode de sous-domaines sans recouvrement sur deux processeurs P_0 et P_1. Nous pouvons choisir d'attribuer les éléments e_0 et e_1 au processeur P_0 et l'élément e_2 au processeur P_1. Cherchons à présent les conditions qui nous permettraient de paralléliser au mieux la discrétisation de la densité de courant $\mathbf{j}$. La première étape consiste à calculer un arbre de facettes pour l'inducteur. Pour paralléliser cette étape, le processeur P_1 ne doit pas attendre de savoir si le processeur P_0 a choisi de placer la facette $f = \{n_3, n_7, n_{11}, n_{15}\}$ dans l'arbre ou le co-arbre. Il faut donc que les deux processeurs "s'entendent" pour la placer dans l'un ou l'autre. Si cette facette est placée dans l'arbre, le flux de $\mathbf{j}$ à travers cette dernière doit être calculé

explicitement. La valeur calculée par le processeur P_0 est alors identique à celle qu'aurait calculée le processeur P_1. Par contre, si nous la plaçons dans le co-arbre, la valeur calculée par le processeur P_0 risque d'être différente de celle qu'aurait calculée le processeur P_1, en raison des différentes erreurs d'arrondis sur les valeurs réelles qui peuvent se produire sur les deux processeurs. Or, il suffit d'un élément de l'inducteur dans lequel la loi de conservation du flux de $\mathbf{j}$ ne soit pas assurée, pour que le gradient conjugué ne converge pas. Il devient alors évident que la facette f doit être placée dans l'arbre. Cette façon de procéder revient à considérer l'inducteur d'origine comme la juxtaposition d'un inducteur composé des éléments e_0 et e_1 et d'un inducteur composé de l'élément e_2. Dès lors, il est possible de construire simultanément deux arbres de facettes, un pour chaque inducteur, ces arbres se raccordant à l'interface entre les sous-domaines pour former une forêt d'arbres de facettes pour l'inducteur d'origine. La différence avec la méthode présentée dans le chapitre 1 réside alors dans le fait qu'un inducteur peut posséder plusieurs points d'entrées, un point par processeur. Cette caractéristique ne constitue pas une violation des hypothèses présentées dans [48] puisque celles-ci précisent que l'arbre de facettes de l'inducteur n'est pas unique. Il peut donc s'agir d'une forêt d'arbres de facettes se raccordant aux interfaces entre les sous-domaines.

La seconde étape consiste à calculer les valeurs à attribuer aux facettes des arbres et des co-arbres. Le calcul relatif aux arbres ne nécessite pas de communications inter-processeurs. Par contre, pour pouvoir calculer les valeurs à attribuer aux facettes de son co-arbre, le processeur P_1 doit connaître la valeur attribuée à la facette $f = \{n_3, n_7, n_{11}, n_{15}\}$ par le processeur P_0. Nous constatons que dans l'exemple de la figure 2.1, le nombre de facettes de d'arbre du processeur P_0 est plus important que celui du processeur P_1, ce qui conduit à un mauvais équilibrage de charge. Pour résoudre ce problème, nous pouvons attribuer les facettes de l'interface entre les sous-domaines à un troisième processeur P_2. De cette manière, le nombre de facettes de l'arbre du processeur P_2 étant très inférieur à ceux des processeurs P_0 et P_1, le calcul relatif à cet arbre se terminera toujours avant celui des processeurs P_0 et P_1. Pour pouvoir calculer la facette $f = \{n_3, n_7, n_{11}, n_{15}\}$, le processeur P_2 doit posséder une copie de l'élément e_1 ou de l'élément e_2. Pour récupérer la valeur attribuée à la facette f par le processeur P_2, les processeurs P_0 et P_1 peuvent se servir de ces copies. Il faut donc que le processeur P_2 dispose d'une copie des éléments e_1 et e_2.

Supposons à présent que nous utilisions une méthode de sous-domaines avec recouvrement. Tirant partie des enseignements précédents, nous pouvons attribuer l'élément e_0 au processeur P_0, l'élément e_1 symbolisant la zone de recouvrement au processeur P_1 et le dernier élément e_2 au processeur P_2. Pour paralléliser l'étape de calcul de l'arbre de facettes, nous devons maintenant considérer l'inducteur initial comme la juxtaposition de trois inducteurs composés respectivement des éléments e_0, e_1 et e_2, les facettes $f_1 = \{n_2, n_6, n_{10}, n_{15}\}$ et $f_2 = \{n_3, n_7, n_{11}, n_{14}\}$ étant placées dans l'arbre de facettes construit par le processeur P_1.

La méthode des éléments finis étant une méthode de discrétisation, elle introduit une erreur dans la solution que seule la finesse du maillage employé permet de minimiser. La discrétisation de la densité de courant ne fait qu'ajouter une

erreur supplémentaire dans cette solution, erreur qui peut être minimisée si le nombre de facettes de l'inducteur est important. Le problème qui se pose alors est celui des méthodes de sous-domaines avec recouvrement, recouvrement symbolisé par l'élément e_1 dans la figure 2.1. Pour conserver un certain taux de parallélisme à ces méthodes, les recouvrements employés sont généralement peu importants. Dès lors, le nombre de facettes de ces zones risque d'être trop faible pour espérer minimiser l'erreur liée à la discrétisation de **j**. En conséquence, afin d'éviter ce problème, nous préférons opter pour les méthodes de sous-domaines sans recouvrement.

Dans la suite de cette section, nous présentons la méthode du complément de Schur [26] qui est le représentant le plus connu des méthodes de sous-domaines sans recouvrement.

2.2.1 Méthode du complément de Schur

Considérons un système d'équations linéaires $Ax = b$ issu de la discrétisation d'équations aux dérivées partielles. Lorsqu'une méthode de sous-domaines sans recouvrement est employée, les inconnues sont divisées en deux catégories : les inconnues internes aux sous-domaines et les inconnues des interfaces entre ces sous-domaines. Les inconnues internes sont numérotées avant celles des interfaces, en parcourant la liste des sous-domaines. Cette numérotation permet de construire la matrice par blocs suivante :

$$A = \begin{pmatrix} B_1 & & & & F_1 \\ & B_2 & & & F_2 \\ & & \ldots & & \ldots \\ & & & B_n & F_n \\ E_1 & E_2 & \ldots & E_n & C \end{pmatrix} \tag{2.2.1}$$

où n désigne le nombre de sous-domaines utilisés.

Le système d'équations initial peut alors se ré-écrire sous la forme :

$$\begin{pmatrix} B & F \\ E & C \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} f \\ g \end{pmatrix} \tag{2.2.2}$$

où x et y sont deux sous-vecteurs contenant respectivement les inconnues internes et celles des interfaces.

Si la matrice B n'est pas singulière, nous pouvons écrire :

$$x = B^{-1}(f - Fy) \tag{2.2.3}$$

puis :

$$(C - EB^{-1}F)y = g - EB^{-1}f \tag{2.2.4}$$

La matrice $S = (C - EB^{-1}F)$ est appelée matrice complément de Schur et possède la même structure et les même propriétés que la matrice A [46]. La méthode du complément de Schur consiste, en partant d'un problème initial relatif à toutes les inconnues des sous-domaines, à se ramener à un problème réduit aux seules inconnues des interfaces entre ces sous-domaines.

À quelques exceptions près [33, 32], la matrice S n'est jamais explicitement calculée, ce qui interdit l'utilisation des méthodes directes. Parmi les méthodes itératives, nous pouvons utiliser la méthode du point fixe ou plus efficacement une méthode de Krylov. Quelle que soit la méthode itérative choisie, nous devons calculer le produit matrice-vecteur $w = Sv$ par la séquence d'opérations suivante :

1. calculer $t = Fv$;
2. résoudre $Bu = t$;
3. calculer $w = Cv - Eu$.

Cette séquence fait intervenir trois produits matrice-vecteur et la résolution d'un système d'équations. Dans le cas des méthodes de Krylov, cette résolution doit être effectuée par une méthode directe ou par une méthode itérative avec une forte précision requise sur la solution, afin de garder la matrice S constante au cours des itérations. Ces contraintes font que les nombres d'itérations nécessaires à la résolution du système réduit et à celle du système global sont sensiblement les mêmes, à l'exception près qu'une itération pour le système réduit coûte plus cher qu'une itération pour le système global. Dans ces conditions, Y. Saad préconise de résoudre le système global en construisant un préconditionneur pour A à partir d'un préconditionneur pour C [57]. Parmi les préconditionneurs que nous pouvons employer, citons les préconditionneurs par blocs de Jacobi, LU et de Gauss-Siedel [56].

Le préconditionneur par blocs de Jacobi est donné par :

$$M = \begin{pmatrix} B & 0 \\ 0 & \widetilde{C} \end{pmatrix} \tag{2.2.5}$$

où $\widetilde{C}$ est une approximation de C. Une résolution triangulaire avec ce préconditionneur nécessite la séquence d'opérations suivante :

1. $x = B^{-1}f$;
2. $y = \widetilde{C}^{-1}g$.

Le préconditionneur LU par blocs est donné par :

$$M = \begin{pmatrix} B & 0 \\ E & \widetilde{C} \end{pmatrix} \begin{pmatrix} I & B^{-1}F \\ 0 & I \end{pmatrix} \tag{2.2.6}$$

et nécessite la séquence d'opérations suivante pour une résolution triangulaire :

1. $x = B^{-1}f$;
2. $y = \widetilde{C}^{-1}(g - Ex)$;
3. $x = x - B^{-1}Fy$.

Le préconditionneur par blocs de Gauss-Siedel est donné par :

$$M = \begin{pmatrix} B & 0 \\ E & \widetilde{C} \end{pmatrix} \tag{2.2.7}$$

et nécessite la séquence d'opérations suivante pour une résolution triangulaire :

1. $x = B^{-1}f$;

2. $y = \widetilde{C}^{-1}(g - Ex)$;

Le choix de la méthode à employer pour une résolution avec la matrice B dépend de son conditionnement. Quand ce dernier est bon, nous pouvons employer une résolution triangulaire avec une factorisation incomplète de B. Si ce conditionnement est un peu moins bon, nous pouvons utiliser une méthode itérative préconditionnée avec une factorisation incomplète de B. Enfin, si ce conditionnement est vraiment mauvais, nous pouvons employer des techniques permettant de construire des approximations creuses plus complexes de B [55, 58] afin de préconditionner une méthode itérative.

2.3 Le code

La méthode du complément de Schur nécessite une certaine régularité des interfaces entre les sous-domaines. Ceci est dû au fait que la matrice S est la forme discrétisée d'un opérateur différentiel assurant des conditions de transmission aux interfaces : l'opérateur de Steklov-Poincaré [63]. Dès lors, il n'est pas possible d'employer des outils de partitionnement automatique des maillages et ce travail fastidieux doit être réalisé par l'utilisateur. Il est rare que nous modélisions un dispositif complet car ce dernier possède généralement des plans de symétrie. Dans ce cas, l'utilisation de conditions de transmission sur ces plans permet de ne modéliser qu'une partie du dispositif. Dès lors, il est logique de penser que le nombre de sous-domaines employés par l'utilisateur sera peu important (de l'ordre de quelques dizaines), que les interfaces entre ces sous-domaines seront des plans et que leur nombre sera également peu important. Dans ces conditions, en supposant que l'utilisateur tentera de répartir au mieux les éléments du maillage entre les sous-domaines, nous pouvons affecter un sous-domaine par processeur et regrouper toutes les interfaces sur un processeur annexe, afin d'obtenir un équilibrage de charge relativement acceptable. Dans le cas général, le processeur dédié aux interfaces devra être sous-employé par rapport aux autres. Les limites de la décomposition du domaine initial seront atteintes lorsque la charge du processeur dédié aux interfaces deviendra pratiquement identique à celle des autres processeurs. Le regroupement de toutes les interfaces sur un même processeur permet de calculer facilement une approximation pour la matrice S, approximation que nous pouvons utiliser pour construire l'un des préconditionneurs par blocs présentés précédemment. Le fait que le nombre de sous-domaines (et donc de blocs) soit peu important et que nous résolvions le système global plutôt que le système réduit, nous délivre de la nécessité d'utiliser le sous-espace V_0 des méthodes de Schwarz et donc du problème lié aux interpolations en dimension trois que nous avons présenté en partie I. Dans ces conditions, le préconditionneur par blocs de Jacobi qui est parfaitement adapté aux machines parallèles, peut s'avérer suffisant.

Ces caractéristiques sont celles de notre nouvelle version Fortran 90 du code. Cette version a été totalement ré-écrite en fonction de la manière dont nous voulions la paralléliser en HPF-Halos. La philosophie de HPF étant de paralléliser un programme existant en s'adaptant à ce dernier, nous avons donc suivi la philosophie exactement inverse. Cette nouvelle application comporte environ 11000

lignes de code. La figure 2.2 présente son algorithme général.

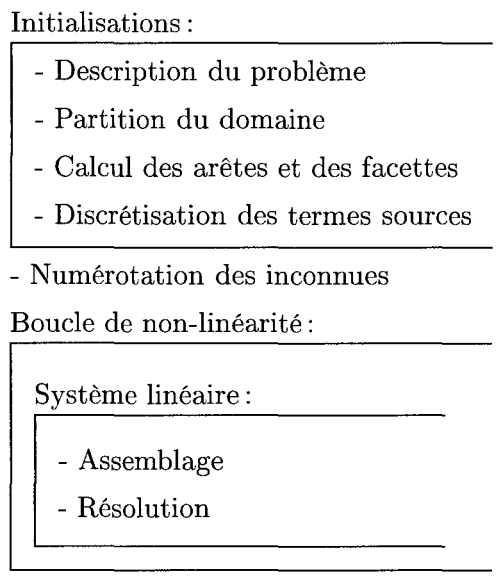

Fig. 2.2 – Algorithme général

L'entité de base de cette nouvelle version est le sous-domaine. Un sous-domaine est un type défini par le programmeur qui comprend :

- des tableaux de nœuds, d'arêtes, de facettes et d'éléments ;
- une matrice carrée et symétrique compressée au format CSR. Cette matrice correspond aux inconnues internes du sous-domaine ;
- une matrice rectangulaire compressée au format CSR ; cette matrice correspond aux couplages entres les inconnues internes et les inconnues des interfaces du sous-domaine.

Certains paramètres tels que les nombres d'arêtes ou de facettes d'un sous-domaine ne peuvent être connus qu'au moment de l'exécution de l'application, ce qui impose la nécessité de surdimensionner les tableaux correspondants dans le type décrivant le sous-domaine. Afin de ne pas introduire de différences entre les sous-domaines et leurs interfaces, ces dernières sont également représentées sous la forme d'un sous-domaine via un système de nœuds et d'éléments "shadow". Cette manière de procéder permet de représenter les sous-domaines et leurs interfaces sous la forme d'un tableau de sous-domaines alloué dynamiquement en fonction des informations données par l'utilisateur dans sa description du problème à traiter. Le sous-domaine correspondant aux interfaces est appelé sous-domaine frontière et occupe toujours la dernière position de ce tableau. Une structure de données globale permet de centraliser toutes les informations relatives aux sous-domaines et au sous-domaine frontière (nombre de nœuds, de nœuds conditionnés, d'arêtes, etc.).

Dans la suite de cette section, nous présentons en détails les caractéristiques de cette nouvelle version.

2.3.1 Description du problème

Comme pour la version précédente, le problème à traiter doit être décrit, mais cette description doit maintenant contenir des informations relatives à la partition du domaine. Dans la version précédente, le maillage était composé de volumes, un matériau étant attribué à chacun d'eux par l'utilisateur. Les inducteurs et les aimants étaient ainsi facilement représentés par de simples listes de matériaux. Lorsque l'utilisateur partitionne son maillage, il définit des volumes englobants qui contiennent les volumes de la version précédente. Ces englobants peuvent alors être qualifiés de sous-domaines physiques et représentés par des listes de matériaux. Par simple fusion de ces listes, il est possible d'associer des sous-domaines physiques pour créer de nouveaux sous-domaines que nous pouvons qualifier de logiques. Dès lors, pour un maillage composé de N sous-domaines physiques, nous pouvons définir toute une suite de partitions en M sous-domaines logiques avec $M \leq N$. Ainsi, dans l'exemple de la figure 2.1, nous pouvons affecter respectivement les matériaux m_0, m_1 et m_2 aux éléments e_0, e_1 et e_2, définir les sous-domaines physiques $d_0 = \{m_0\}$, $d_1 = \{m_1\}$ et $d_2 = \{m_2\}$, puis créer les partitions logiques $D_0 = \{\{m_0, m_1\}, \{m_2\}\}$, $D_1 = \{\{m_0\}, \{m_1, m_2\}\}$ et $D_2 = \{\{m_0\}, \{m_1\}, \{m_2\}\}$.

L'utilisation de nœuds et d'éléments "shadow" impose de décrire explicitement chaque interface et de lui attribuer deux identificateurs. Le premier identificateur id_1 est un numéro unique attribué à chaque interface et qui permet d'établir des règles de propriété (qui calcule quoi) pour le calcul des arêtes et des facettes du maillage. Le second identificateur id_2 est un numéro d'arbre unique qui permet d'éviter les cycles lors de la construction d'une forêt d'arbres d'arêtes pour ce maillage. Ainsi, deux interfaces disjointes ont des numéros d'arbres différents. Les numéros id_2 sont identiques lorsque les interfaces sont en contact via les intersections d'interfaces ou via l'ajout d'une arête entre ces deux interfaces durant la phase de construction de la forêt. Si nous appliquons la partition D_2 à l'exemple de la figure 2.1, nous décrirons explicitement les interfaces $B_0 = \{\{m_0\}, \{m_1\}\}$ et $B_1 = \{\{m_1\}, \{m_2\}\}$, puis nous pourrons leur attribuer respectivement les couples d'identificateurs $(id_1 = 0, id_2 = 0)$ et $(id_1 = 1, id_2 = 1)$. Cela signifie d'une part (identificateur id_1) que l'arête $\{n_2, n_{10}\}$ doit être calculée par le sous-domaine frontière puisque ses nœuds appartiennent à la même interface, mais que l'arête $\{n_2, n_3\}$ doit être calculée par le sous-domaine $\{m_1\}$. Cela signifie d'autre part (identificateur id_2) que si aucune arête entre les interfaces B_0 et B_1 n'a encore été ajoutée dans l'arbre d'arêtes construit par le sous-domaine m_1, alors nous pouvons y ajouter l'arête $\{n_2, n_3\}$. Une fois cette opération effectuée, les arêtes des deux interfaces seront en contact et porteront de fait le même identificateur id_2.

2.3.2 Partition du maillage

La partition du maillage est effectuée en fonction des informations données par l'utilisateur dans sa description du problème à traiter. Cette partition comprend cinq étapes. Pour les illustrer, nous allons appliquer la partition D_2 au maillage de la figure 2.1. Nous supposerons que le nombre maximum N de nœuds

par sous-domaine est fixé à huit et que le nombre maximum E d'éléments par sous-domaine est fixé à trois.

Tant que le fichier contenant le maillage n'a pas été lu dans son intégralité, il est impossible de commencer le partitionnement. La première étape consiste donc à stocker les nœuds et les éléments lus en fonction des constantes N et E. La figure 2.3 présente les résultats de cette première étape.

sous-domaine	$\{m_0\}$	$\{m_1\}$	$\{m_2\}$	frontière
nœuds	n_1, n_2, n_3, n_4 n_5, n_6, n_7, n_8	$n_9, n_{10}, n_{11}, n_{12}$ $n_{13}, n_{14}, n_{15}, n_{16}$		
éléments	e_0, e_1, e_2			

Fig. 2.3 – Étape 1 : stockage des nœuds et des éléments du maillage

La seconde étape consiste à répartir les éléments entre les sous-domaines et à fusionner les volumes de même perméabilité magnétique à l'intérieur de chaque sous-domaine. Cette fusion permettra, par la suite, d'obtenir un meilleur conditionnement pour chaque bloc diagonal de la matrice du système d'équations global. Lors de la répartition, chaque élément se voit attribuer un numéro de sous-domaine. La figure 2.4 présente les résultats de cette seconde étape.

sous-domaine	$\{m_0\}$	$\{m_1\}$	$\{m_2\}$	frontière
nœuds	n_1, n_2, n_3, n_4 n_5, n_6, n_7, n_8	$n_9, n_{10}, n_{11}, n_{12}$ $n_{13}, n_{14}, n_{15}, n_{16}$		
éléments	e_0	e_1	e_2	

Fig. 2.4 – Étape 2 : répartition des éléments

La troisième étape consiste à répartir les nœuds entre les sous-domaines et le sous-domaine frontière. Pour connaître le propriétaire d'un nœud, nous utilisons la liste des éléments auxquels ce nœud est connecté pour dresser celle des sous-domaines d'appartenance. Si cette dernière comprend au moins deux sous-domaines, il s'agit d'un nœud du sous-domaine frontière. Dans le cas contraire, il s'agit d'un nœud interne à un sous-domaine. La figure 2.5 présente les résultats de cette troisième étape.

sous-domaine	$\{m_0\}$	$\{m_1\}$	$\{m_2\}$	frontière
nœuds	n_1, n_5, n_9, n_{13}		n_4, n_8, n_{12}, n_{16}	$n_2, n_3, n_6, n_7,$ $n_{10}, n_{11}, n_{14}, n_{15}$
éléments	e_0	e_1	e_2	

Fig. 2.5 – Étape 3 : répartition des nœuds

La quatrième étape consiste à créer les nœuds "shadow". Pour cela, les nœuds du sous-domaine frontière sont répliqués dans les listes de nœuds des sous-domaines concernés, derrière les nœuds originaux de ces derniers. La figure 2.6 présente les résultats de cette quatrième étape.

sous-domaine	$\{m_0\}$	$\{m_1\}$	$\{m_2\}$	frontière
nœuds	$n_1, n_5, n_9, n_{13}, n_2', n_{10}', n_6', n_{14}'$	$n_2', n_3', n_6', n_7', n_{10}', n_{11}', n_{14}', n_{15}'$	$n_4, n_8, n_{12}, n_{16}', n_3', n_7', n_{11}', n_{15}'$	n_2, n_3, n_6 n_7, n_{10}, n_{11}, n_{14}, n_{15}
éléments	e_0	e_1	e_2	

Fig. 2.6 – Étape 4 : création des nœuds "shadow"

La cinquième et dernière étape consiste à créer les éléments "shadow". Pour cela, nous répliquons les éléments des sous-domaines dont au moins un nœud appartient au sous-domaine frontière, dans la liste des éléments de ce dernier. La figure 2.7 présente les résultats de cette cinquième étape.

sous-domaine	$\{m_0\}$	$\{m_1\}$	$\{m_2\}$	frontière
nœuds	$n_1, n_5, n_9, n_{13}, n_2', n_{10}', n_6', n_{14}'$	$n_2', n_3', n_6', n_7', n_{10}', n_{11}', n_{14}', n_{15}'$	$n_4, n_8, n_{12}, n_{16}', n_3', n_7', n_{11}', n_{15}'$	n_2, n_3, n_6 n_7, n_{10}, n_{11}, n_{14}, n_{15}
éléments	e_0	e_1	e_2	e_0', e_1', e_2'

Fig. 2.7 – Étape 5 : création des éléments "shadow"

Les nœuds et les éléments sont étroitement interconnectés via les listes d'indices d'éléments et de nœuds qui leur sont respectivement associées. Ces listes sont initialisées dès la lecture du maillage et sont constamment mises à jour au cours des cinq étapes de sa partition. Ainsi, à l'intérieur des sous-domaines et du sous-domaine frontière, chaque nœud dispose d'une liste d'indices L_1 référençant les éléments auxquels il est connecté. Selon la valeur de ces indices, nous savons si les éléments référencés sont locaux ou non-locaux au sous-domaine ou au sous-domaine frontière concerné (dans notre cas, seuls les nœuds du sous-domaine frontière peuvent référencer des éléments non-locaux). Pour chaque élément non-local référencé, un élément "shadow" correspondant est créé puis ajouté à la fin de la liste des éléments du sous-domaine ou du sous-domaine frontière concerné. Pour permettre sa manipulation, nous associons une seconde liste d'indices au nœud : celle des éléments "shadow" auxquels ce dernier est connecté. Ainsi, si $L_1(i)$ référence un élément non-local, $L_2(i)$ référence l'élément "shadow" correspondant. La figure 2.8 permet d'illustrer ce mécanisme.

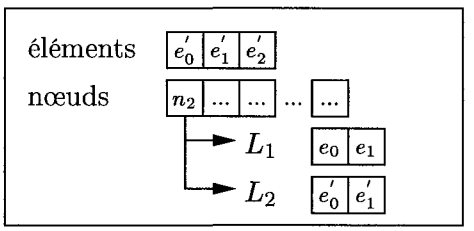

Fig. 2.8 – Éléments "shadow" dans le sous-domaine frontière

Dans cette figure, le nœud n_2 du sous-domaine frontière référence deux éléments non-locaux : l'élément e_0 qui appartient au sous-domaine $\{m_0\}$ et l'élément e_1 qui appartient au sous-domaine $\{m_1\}$. Deux éléments "shadow" correspondants e_0' et e_1' sont alors créés et ajoutés dans la liste des éléments du sous-domaine frontière. Pour permettre leur manipulation par le nœud n_2, celui-ci se voit doté d'une seconde liste d'indices L_2.

Les éléments disposent également d'une liste d'indices L_1 référençant les nœuds qui leur sont connectés. La valeur de ces indices permet de savoir si les nœuds référencés sont locaux ou non. Pour chaque nœud non-local référencé, un nœud "shadow" est créé et ajouté à la fin de la liste des nœuds du sous-domaine ou du sous-domaine frontière concerné (dans notre cas, seuls les éléments des sous-domaines peuvent référencer des nœuds non-locaux). Pour permettre leur manipulation, nous associons une seconde liste d'indices à l'élément : celles des nœuds "shadow" qui lui sont connectés. La figure 2.9 permet d'illustrer ce mécanisme.

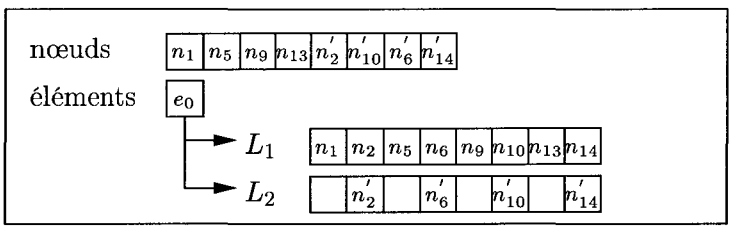

Fig. 2.9 – Nœuds "shadow" dans le sous-domaine $\{m_0\}$

Dans cette figure, l'élément e_0 du sous-domaine $\{m_0\}$ référence quatre nœuds non-locaux appartenant au sous-domaine frontière. Quatre nœuds "shadow" correspondants sont alors créés et ajoutés dans la liste des nœuds du sous-domaine $\{m_0\}$. Pour permettre leur manipulation par l'élément e_0, celui-ci se voit doté de la nouvelle liste d'indices L_2.

Dans notre future version HPF-Halos du code, le maillage devra être lu puis partitionné par un unique processeur. Chaque processeur recevra ensuite le sous-domaine ou le sous-domaine frontière qui le concerne. Cette opération nécessitera donc une recomposition du tableau contenant les sous-domaines et le sous-domaine frontière sur le processeur chargé de lire et de partitionner le maillage, puis une diffusion vers tous les autres processeurs. Afin de réduire les volumes de données échangées au cours de ces opérations, nous avons fractionné le type permettant de représenter les sous-domaines en deux types complémentaires. Le premier type ne contient que les tableaux de nœuds et d'éléments du sous-domaine. Le second type contient la matrice carrée relative à ses inconnues internes et la matrice rectangulaire relative aux couplages entre ses inconnues internes et ses inconnues des interfaces. Le fractionnement du type d'origine entraîne celui du tableau contenant les sous-domaines et le sous-domaine frontière en deux tableaux complémentaires alloués dynamiquement. De cette façon, les opérations de recomposition et de diffusion ne concerneront alors plus que le tableau relatif aux nœuds et aux éléments des sous-domaines et du sous-domaine

frontière.

2.3.3 Calcul des arêtes et des facettes

Lorsque les nœuds et les éléments "shadow" sont créés, les sous-domaines et le sous-domaine frontière peuvent calculer leurs arêtes et leurs facettes indépendamment les uns des autres. Les listes de nœuds correspondantes aux arêtes et aux facettes calculées ne contiennent que des nœuds locaux : il peut donc s'agir de nœuds originaux ou de nœuds "shadow". Les facettes sont connectées à leurs arêtes et aux éléments qui les partagent via des listes d'indices analogues aux listes d'indices L_1 des nœuds et des éléments. Là encore, tous les éléments et arêtes référencés sont locaux.

2.3.4 Discrétisation des termes sources

La densité de courant $\mathbf{j}$ et le champ source $\mathbf{h_s}$ sont les termes sources relatifs aux inducteurs. Le principe que nous utilisons pour les discrétiser a été abordé de manière assez générale dans la section 2.2 lors du choix d'une méthode de Schwarz qui convienne à notre application. Ce principe fait intervenir des communications d'assemblage que nous présenterons ultérieurement. Afin de l'illustrer, nous allons considérer l'exemple de la figure 2.10.

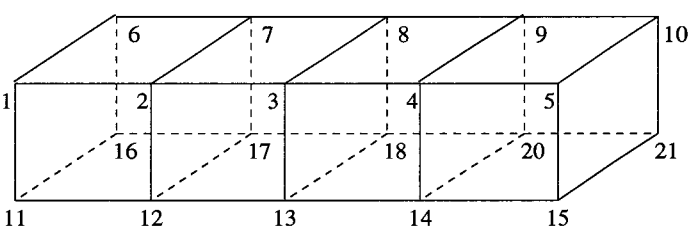

Fig. 2.10 – Partition d'un inducteur

Cette figure représente un inducteur composé de quatre éléments : l'élément $e_0 = \{n_1, n_2, n_6, n_7, n_{11}, n_{12}, n_{16}, n_{17}\}$ auquel est associé le milieu m_0, l'élément $e_1 = \{n_2, n_3, n_7, n_8, n_{12}, n_{13}, n_{17}, n_{18}\}$ auquel est associé le milieu m_1, l'élément $e_2 = \{n_3, n_4, n_8, n_9, n_{13}, n_{14}, n_{18}, n_{20}\}$ auquel est associé le milieu m_2 et l'élément $e_3 = \{n_4, n_5, n_9, n_{10}, n_{14}, n_{15}, n_{20}, n_{21}\}$ auquel est associé le milieu m_3. À cet inducteur, nous appliquons la partition $D = \{\{m_0, m_1\}, \{m_2, m_3\}\}$.

Densité de courant

La densité de courant $\mathbf{j}$ doit être discrétisée dans l'espace $\mathbf{W^2}$ des éléments finis de facettes et être à divergence nulle. Le principe que nous utilisons pour la discrétiser consiste non plus à construire un arbre de facettes pour l'inducteur comme nous le faisions dans la version précédente du code, mais une forêt d'arbres (un arbre pour chaque sous-domaine et le sous-domaine frontière) se raccordant aux interfaces entre les sous-domaines. Toutes les facettes des interfaces qui appartiennent à l'inducteur sont systématiquement placées dans l'arbre de facettes construit par le sous-domaine frontière.

La discrétisation de la densité de courant s'effectue en trois étapes. La première étape consiste, pour les sous-domaines et le sous-domaine frontière, à calculer leurs propres arbres de facettes et le flux de **j** au travers de ces dernières, indépendamment les uns des autres. La figure 2.11 présente une forêt composée de trois arbres et de deux co-arbres (en gras) de facettes calculée pour l'inducteur à l'issue de cette étape.

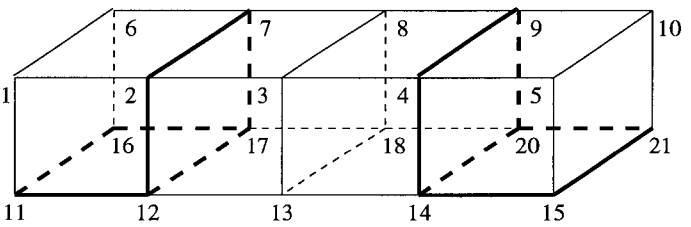

Fig. 2.11 – Forêt d'arbres et de co-arbres (en gras) de facettes

La seconde étape consiste, pour le sous-domaine frontière, à envoyer aux sous-domaines concernés les valeurs du flux de **j** calculées pour les facettes des interfaces ($f = \{n_3, n_8, n_{13}\ n_{18}\}$). Cet envoi est effectué via les communications d'assemblage. La troisième et dernière étape consiste, pour les sous-domaines, à calculer les valeurs du flux de **j** au travers des facettes de leurs co-arbres, indépendamment les uns des autres.

Champ source

Le champ source $\mathbf{h_s}$ doit être discrétisé dans l'espace $\mathbf{W^1}$ des éléments finis d'arêtes, de telle manière que le théorème d'Ampère soit vérifié sur toutes les facettes du maillage. Le principe que nous utilisons pour le discrétiser est sensiblement le même que celui employé pour la discrétisation de la densité de courant. Nous disons sensiblement car contrairement à la densité de courant, il n'est pas possible ici de placer toutes les arêtes des interfaces dans l'arbre d'arêtes construit par le sous-domaine frontière. Ainsi, dans l'exemple de la figure 2.10, nous ne pouvons placer qu'un maximum de trois arêtes de la facette $f = \{n_3, n_8, n_{13}\ n_{18}\}$ dans l'arbre d'arêtes construit par le sous-domaine frontière, la dernière devant être placée dans son co-arbre pour que le théorème d'Ampère soit vérifié sur toutes les facettes des interfaces. Il est cependant possible que les sous-domaines et le sous-domaine frontière calculent leurs propres arbres d'arêtes et la circulation de $\mathbf{h_s}$ sur les arêtes des co-arbres, indépendamment les uns des autres. Pour cela, les sous-domaines doivent supposer que toutes les arêtes des interfaces sont placées dans l'arbre d'arêtes construit par le sous-domaine frontière. Pour illustrer ce mécanisme, supposons que le sous-domaine frontière ait décidé de placer les arêtes $\{n_3, n_8\}$, $\{n_3, n_{13}\}$ et $\{n_{13}, n_{18}\}$ de la facette f dans son arbre d'arêtes, la dernière arête $\{n_8, n_{18}\}$ étant placée dans son co-arbre afin que le théorème d'Ampère soit vérifié. Considérons à présent la facette $f' = \{n_7, n_8, n_{17}, n_{18}\}$ du sous-domaine $\{m_0, m_1\}$. Ce dernier peut décider de placer les arêtes $\{n_3, n_7\}$ et $\{n_7, n_8\}$ dans son arbre d'arêtes. Il ne peut faire de même pour l'arête $\{n_{17}, n_{18}\}$ car l'identificateur id_2 lui signale-

rait alors l'existence d'un cycle, toutes les arêtes des interfaces étant considérées comme faisant partie de l'arbre construit par le sous-domaine frontière. Ainsi, quelles que soient les valeurs calculées par le sous-domaine frontière pour les arêtes des interfaces, il reste toujours au moins une arête de la facette f' pour que le théorème d'Ampère soit vérifié.

Comme pour la discrétisation de la densité de courant, celle du champ source peut être effectuée en trois étapes. La première étape consiste, pour les sous-domaines et le sous-domaine frontière, à calculer une forêt d'arbres d'arêtes (un arbre pour chaque sous-domaine et le sous-domaine frontière) puis la circulation de $\mathbf{h_s}$ sur les arêtes des co-arbres, indépendamment les uns des autres. Pour cela, les sous-domaines doivent supposer que toutes les arêtes des interfaces sont placées dans l'arbre d'arêtes construit par le sous-domaine frontière (l'identificateur id_2 leur permet de calculer des arbres en fonction de cette hypothèse). Durant cette étape, le sous-domaine frontière doit également appliquer le théorème d'Ampère aux arêtes de son arbre. La seconde étape consiste, pour le sous-domaine frontière, à envoyer les valeurs calculées pour les arêtes des interfaces aux sous-domaines concernés. Cet envoi est effectué via les communications d'assemblage. La troisième et dernière étape consiste, pour les sous-domaines, à utiliser les valeurs transmises par le sous-domaine frontière pour appliquer le théorème d'Ampère aux arêtes de leurs arbres, indépendamment les uns des autres.

La figure 2.12 présente une forêt composée de trois arbres (en gras) et de trois co-arbres d'arêtes calculée pour le maillage de la figure 2.10.

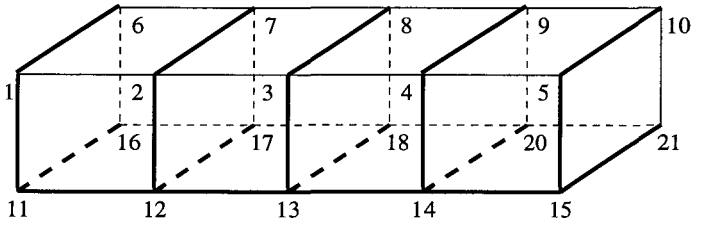

Fig. 2.12 – Forêt d'arbres (en gras) et de co-arbres d'arêtes

2.3.5 Numérotation des inconnues

En utilisant la structure de données globale contenant les renseignements sur les sous-domaines et le sous-domaine frontière, ces derniers peuvent numéroter leurs inconnues indépendamment les uns des autres. Ainsi, si la formulation magnétostatique employée est la formulation en potentiel scalaire magnétique, il suffit, pour le sous-domaine ou le sous-domaine frontière occupant la nième position dans le tableau contenant les sous-domaines, de calculer la quantité :

$$s = \sum_{i=1}^{n-1} (n^i - n_c^i) \tag{2.3.1}$$

où n^i et n_c^i désignent respectivement le nombre de nœuds originaux et le nombre de nœuds originaux conditionnés du ième sous-domaine. Le premier numéro

d'inconnue à attribuer est alors égal à $s + 1$. Si la formulation magnétostatique employée est la formulation en potentiel vecteur magnétique, les quantités n^i et n_c^i font alors référence aux arêtes du ième sous-domaine.

La numérotation des inconnues s'effectue en deux étapes. La première étape consiste, pour les sous-domaines et le sous-domaine frontière, à numéroter leurs inconnues indépendamment les uns des autres en utilisant la technique présentée ci-dessus. La seconde étape consiste, pour les sous-domaines, à échanger avec le sous-domaine frontière les numéros d'inconnues attribués aux nœuds ou au arêtes des éléments auxquels sont associés des éléments "shadow". Cet échange est réalisé via les communications d'assemblage.

2.3.6 Assemblage

Une fois les numéros d'inconnues relatifs aux éléments "shadow" échangés, les sous-domaines et le sous-domaine frontière peuvent construire la partie du système d'équations qui les concerne (structure CSR et assemblage), indépendamment les uns des autres. La matrice du système d'équations global étant symétrique, nous effectuons les choix suivants :

- chaque sous-domaine construit la partie triangulaire inférieure de son bloc diagonal B_i (inconnues internes) ainsi que son bloc d'interface F_i (couplage avec les inconnues des interfaces). Ces deux blocs sont construits séparément et constituent de fait des matrices indépendantes avec leurs propres structures CSR ;
- le sous-domaine frontière ne construit que la partie triangulaire inférieure de son bloc diagonal C (inconnues des interfaces).

À l'exception des règles de propriété (qui calcule quoi) concernant les entrées de la matrice du système d'équations, les algorithmes utilisés sont en tout point semblables à ceux de la version précédente du code. La figure 2.13 illustre cette construction dans le cas d'une partition en deux sous-domaines.

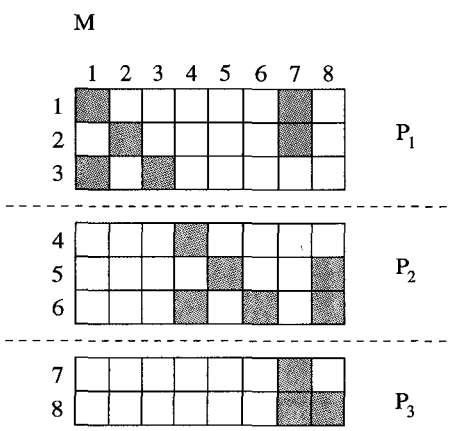

Fig. 2.13 – Partition en 2 sous-domaines

2.3.7 Résolution

Comme nous l'avons déjà dit dans le chapitre 1, la méthode du gradient conjugué préconditionné fait intervenir trois types d'opérations :

- des produits scalaires ;
- des produits matrice-vecteur $b = Mx$ où M désigne la matrice du système d'équations ;
- des résolutions triangulaires $Nx = b$ où N désigne la matrice de préconditionnement.

Contrairement à la version précédente du code, les matrices M et N disposent maintenant de structures par blocs et tout vecteur x se décompose sous la forme de deux sous-vecteurs x_1 et x_2 contenant respectivement les inconnues internes aux sous-domaines et les inconnues des interfaces entre ces sous-domaines. Nous pouvons alors écrire le produit matrice-vecteur $b = Mx$ sous la forme :

$$\begin{pmatrix} b_1 \\ b_2 \end{pmatrix} = \begin{pmatrix} B & F \\ E & C \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \tag{2.3.2}$$

ce qui nécessite la séquence d'opérations :

1. $b_1 = Bx_1 + Fx_2$;
2. $b_2 = Ex_1 + Cx_2$.

La matrice M étant symétrique, nous avons choisi de la stocker sous la forme présentée dans la figure 2.13. Dès lors, nous devons remplacer la séquence précédente par :

1. $b_1 = Bx_1 + Fx_2$;
2. $b_1' = F^T x_1$;
3. $b_2 = b_1' + Cx_2$.

Ce remplacement doit également être effectué pour les séquences d'opérations liées aux préconditionneurs par blocs que nous avons présentées dans la section 2.2. Ces séquences montrent que les communications liées à la résolution sont des communications entre les sous-domaines et le sous-domaine frontière et que les blocs d'interface F_i en sont la cause.

Les approximations que nous utilisons pour les matrices B_i et C sont des factorisations incomplètes de Crout. Des trois préconditionneurs par blocs présentés dans la section 2.2, nous avons écarté celui de Gauss-Siedel car les nombres d'itérations obtenus avec ce dernier pour tous les exemples traités étaient sensiblement les mêmes que ceux obtenus avec le préconditionneur par blocs de Jacobi. Les algorithmes utilisés par les sous-domaines et le sous-domaine frontière pour calculer les factorisations incomplètes de Crout, pour effectuer les résolutions triangulaires avec ces dernières et les produits matrice-vecteur avec les blocs diagonaux B_i, sont en tout point semblables à ceux de la version précédente du code.

2.3.8 Communications

Les communications entre les sous-domaines et le sous-domaine frontière sont de deux types : les communications d'assemblage et les communications de

résolution. Les première sont liées aux élément "shadow". Les secondes sont liées aux blocs d'interface de la matrice du système d'équations et éventuellement à ceux de la matrice de préconditionnement.

Communications d'assemblage

Les communications d'assemblage sont liées aux éléments "shadow". Elles concernent les circulations du champ source calculées sur leurs arêtes, les flux de la densité de courant calculés au travers de leurs facettes, ainsi que les numéros d'inconnues attribués à leurs nœuds ou à leurs arêtes. Ces arêtes et ces facettes étant orientées, un signe est associé à chaque circulation, à chaque flux et à chaque inconnue si cette dernière est relative à une arête.

Toutes ces informations sont regroupées au sein d'un type "message" défini par le programmeur. Ce type est logiquement calqué sur celui permettant de décrire les éléments. Un type "tableau de messages" est également défini par le programmeur pour contenir un tableau de messages surdimensionné. Son utilité sera précisée ultérieurement. À chaque sous-domaine et au sous-domaine frontière, nous associons deux tableaux de messages : un pour les messages à envoyer (*send*) et un pour les messages reçus (*received*). L'algorithme que nous avons employé pour créer les éléments "shadow", fait qu'à chaque sous-domaine est associée une suite d'éléments "shadow" consécutifs dans la liste des éléments du sous-domaine frontière. Dès lors, à chaque sous-domaine est également associée une suite de messages consécutifs dans les tableaux *send* et *received*. Le fait de définir le type "tableau de messages" nous permet d'attribuer la même position dans ces tableaux à l'élément d'un sous-domaine et à son élément "shadow" associé dans le sous-domaine frontière. Le calcul de ces positions est effectué par les sous-domaines et le sous-domaine frontière, indépendamment les uns des autres, en consultant la structure de données globale contenant les informations relatives à ces derniers. La figure 2.14 permet d'illustrer ce mécanisme dans le cas du maillage de la figure 2.1 partitionné en trois sous-domaines.

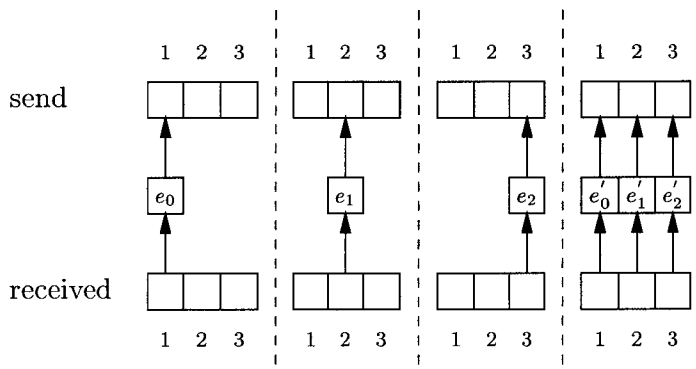

Fig. 2.14 – Tableaux de messages

Dans cette figure, un tableau de messages à envoyer (*send*) et un tableau de messages reçus (*received*) sont associés à chaque sous-domaine et au sous-domaine frontière. Ces tableaux sont surdimensionnés pour contenir trois mes-

sages, ce qui correspond (pour les besoins de l'exemple) au nombre d'éléments "shadow" du sous-domaine frontière. À chaque sous-domaine correspond une suite de messages consécutifs dans ces tableaux. Dans notre cas, cette suite est réduite à un message. Le fait de définir un type "tableau de messages" permet d'attribuer la même position dans ces tableaux ($p = 3$) à l'élément e_2 et à son élément "shadow" associé e_2'.

Grâce aux tableaux de messages qui leur sont associés, les sous-domaines et le sous-domaine frontière peuvent préparer les messages à envoyer et consulter les messages reçus indépendamment les uns des autres. La préparation des messages à envoyer consiste à collecter les informations requises par le programmeur via un "bon de commande" et à les placer dans le tableau correspondant.

Dans notre future version HPF-Halos, nous voulons que le mécanisme de délivrance des messages soit basé sur Halos et qu'il soit implanté dans une zone bien précise et limitée du code. Pour cela, dans notre nouvelle version Fortran 90, nous créons dynamiquement un tableau unidimensionnel *tmp* pouvant contenir $2n$ messages, n désignant le nombre d'éléments "shadow" du sous-domaine frontière, ainsi qu'un tableau unidimensionnel *itmp* pouvant contenir $2n$ indices. Ces deux tableaux seront alignés et feront l'objet d'une directive de répartition par blocs généralisée dans notre future version HPF-Halos. Pour chaque sous-domaine, la taille du bloc sera égale au nombre de ses éléments possédant un élément "shadow" associé dans le sous-domaine frontière. Pour ce dernier, la taille du bloc sera égale au nombre de ses éléments "shadow". Dès lors, le fait d'avoir défini le type "tableau de messages" permettra aux sous-domaines et au sous-domaine frontière d'initialiser le tableau *itmp*, indépendamment les uns des autres. En effet, pour chaque élément d'un sous-domaine auquel est associée la position p dans ses tableaux *send* et *received*, la valeur calculée pour $itmp(p)$ est $p + n$. Pour chaque élément "shadow" du sous-domaine frontière auquel est associée la position p dans ses tableaux *send* et *received*, la valeur calculée pour $itmp(p + n)$ est également $p + n$. Ainsi, pour chaque élément d'un sous-domaine auquel est associée la position p dans ses tableaux *send* et *received*, la valeur correspondante dans $itmp(p)$ peut être calculée avec la seule connaissance du nombre d'éléments "shadow" du sous-domaine frontière. La figure 2.15 permet d'illustrer ce mécanisme pour l'exemple de la figure précédente.

Dans cette figure, les sous-domaines et le sous-domaine frontière accèdent au tableau *tmp* via le tableau d'indices *itmp*. Les communications entre les sous-domaines et le sous-domaine frontière peuvent être mono ou bi-directionnelles. Les communications bi-directionnelles sont traitées comme des séquences de deux communications mono-directionnelles. Une commmunication mono-directionnelle comporte trois étapes. Ainsi, dans le cas d'une communication depuis les sous-domaines vers le sous-domaine frontière, la première étape consiste, pour les sous-domaines, à préparer les messages à envoyer en collectant les informations requises et en les déposant dans les tableaux *send* qui leur sont associés, indépendamment les uns des autres. La seconde étape consiste, pour le mécanisme de délivrance des messages, à prélever les messages à envoyer dans les tableaux *send* associés aux sous-domaines et à les déposer dans le tableau *tmp* via le tableau d'indices *itmp*. La troisième étape consiste, pour le sous-domaine frontière, à prélever les messages stockés dans le tableau *tmp* via le

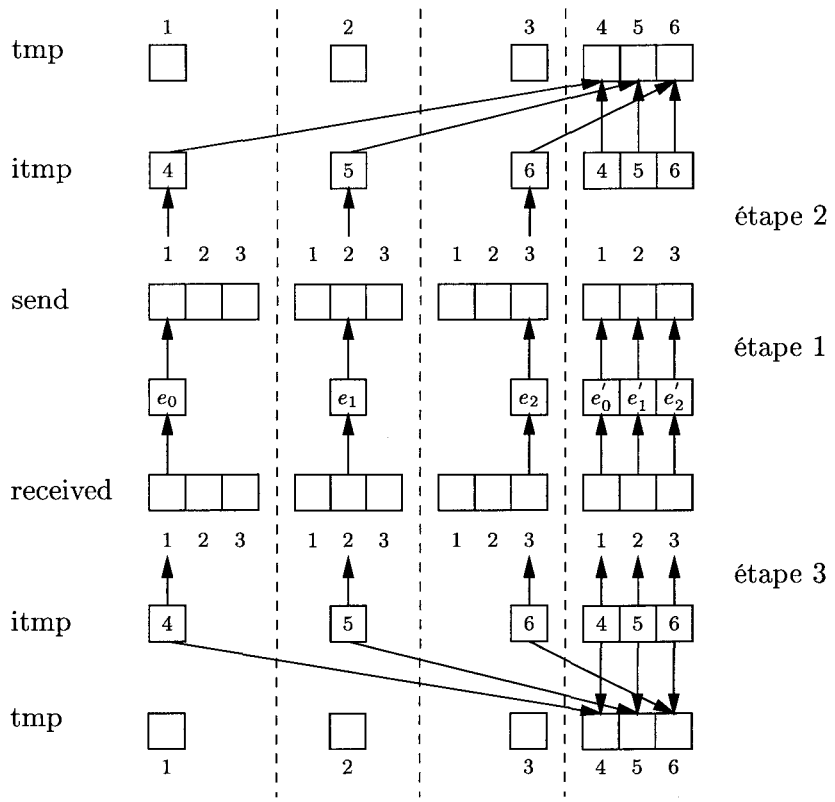

Fig. 2.15 – Mécanisme d'échange des messages

tableau d'indices *itmp* et à les déposer dans le tableau *received* qui lui est associé. Les communications depuis le sous-domaine frontière vers les sous-domaines utilisent les opérations inverses.

Du tableau *tmp* que nous déclarons, seule la partie attribuée au sous-domaine frontière est réellement utilisée, ce qui constitue une perte de mémoire. Cependant, l'utilisation de Halos nous impose de procéder de la sorte. Nous aurions également pu, afin d'optimiser les communications, créer un type de message spécifique à chaque étape du code: discrétisation de la densité de courant, discrétisation du champ source et échange des numéros d'inconnues. Cependant, nous avons estimé d'une part que ces communications étaient trop peu nombreuses pour justifier une démultiplication des tableaux *tmp* et *itmp*. Nous avons estimé d'autre part que le fait de réunir toutes les informations au sein d'un même type, permettait au programmeur d'ajouter facilement de nouvelles informations à transporter, le seul travail requis étant d'implémenter la collecte de ces dernières.

Communications de résolution

Les communications de résolution sont liées aux blocs d'interface de la matrice du système d'équations et éventuellement à ceux de la matrice de préconditionnement. Pour les gérer, un type "sous-vecteurs" est défini par le programmeur pour contenir deux sous-vecteurs surdimensionnés. Son utilité sera précisée ultérieurement. Le premier sous-vecteur est relatif aux inconnues internes du sous-domaine. Le second est relatif à ses inconnues des interfaces. Dans le cas du sous-domaine frontière, seul le premier sous-vecteur est utilisé puisque nous considérons que les inconnues des interfaces sont les inconnues internes de ce dernier. Lorsqu'un produit matrice-vecteur ou une résolution triangulaire avec la matrice de préconditionnement doit être effectué, la séquence d'opérations exécutée est la suivante:

1. pré-chargement;
2. calcul avec les sous-vecteurs;
3. post-stockage.

Les étapes de pré-chargement et de post-stockage permettent de ne jamais travailler directement sur un vecteur utilisé par le gradient conjugué. L'objectif recherché est, comme pour les communications d'assemblage, de contrôler et de centraliser toutes les communications de résolution entre les sous-domaines et le sous-domaine frontière en une zone bien précise et limitée du code. Dans le cas d'un produit matrice-vecteur $b = Mx$, l'étape de pré-chargement consiste, pour les sous-domaines et le sous-domaine frontière, à prélever les entrées du vecteur x qui les concernent et à les stocker dans les sous-vecteurs x_1 et x_2 qui leur sont associés. La figure 2.16 permet d'illustrer ce mécanisme.

L'étape de calcul consiste, pour les sous-domaines et le sous-domaine frontière, à effectuer la partie du calcul qui les concerne en ne prenant comme entrées et comme sorties que les sous-vecteurs x_1 et x_2 qui leur sont associés. Cette étape ne nécessite donc pas de communications entre les sous-domaines et le sous-domaine frontière. Les blocs diagonaux B_i et C ainsi que les blocs d'interface F_i ayant été construits séparément, chacune de ces matrices est indépendante,

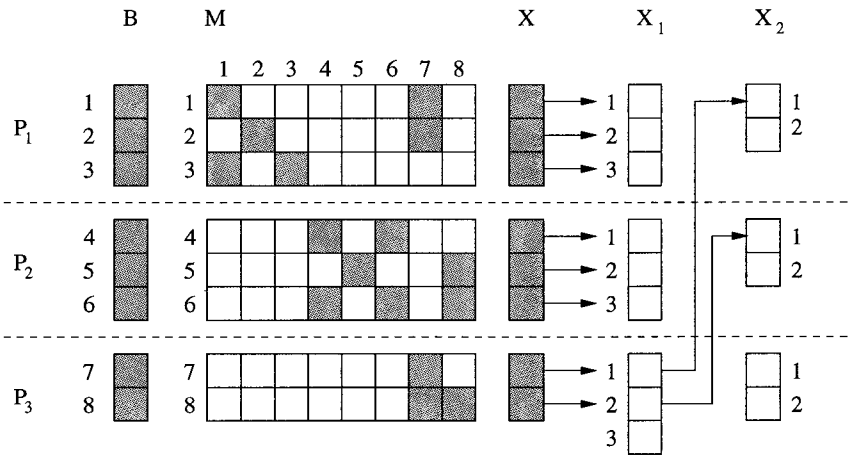

Fig. 2.16 – Pré-chargement

possède sa propre structure CSR et donc son propre tableau d'indices de colonnes jA. Dès lors, les premières lignes et colonnes de ces matrices portent le numéro un, ce qui fait que les tableaux jA ne peuvent être utilisés pour accéder directement aux vecteurs utilisés par le gradient conjugué. La définition du type "sous-vecteurs" permet de résoudre ce problème, ainsi que le montre la figure 2.16.

L'étape de post-stockage consiste, pour les sous-domaines et le sous-domaine frontière, à mettre à jour un vecteur utilisé par le gradient conjugué à partir des sous-vecteurs x_1 et x_2 qui leur sont associés. Ces derniers peuvent éventuellement être combinés avant la mise à jour. Ainsi, dans le cas du produit matrice-vecteur $b = Mx$, chaque sous-domaine utilise son sous-vecteur x_1 pour stocker la partie du produit qui le concerne (bloc diagonal B_i et bloc d'interface F_i) et son sous-vecteur x_2 pour stocker la partie du produit qui concerne le sous-domaine frontière (transposée du bloc d'interface F_i). Le sous-domaine frontière, quant à lui, utilise son sous-vecteur x_1 pour stocker la partie du produit relative au bloc diagonal C. Dès lors, pour obtenir le résultat final, les sous-vecteur x_2 des sous-domaines et le sous-vecteur x_1 du sous-domaine frontière doivent être combinés. Cette opération est illustrée par la figure 2.17.

La manière de combiner les sous-vecteurs étant dépendante de l'opération à mener, un post-stockage spécifique doit être associé à chacune de ces opérations, ce qui n'est pas le cas du pré-chargement qui, quant à lui, est général. Dans cette nouvelle version du code, seuls les produits scalaires agissent directement sur les vecteurs utilisés par le gradient conjugué. Les autres opérations sont basées sur des étapes de pré-chargement et de post-stockage. À chacune de ces opérations est associé un post-stockage spécifique.

Comme pour les communications d'assemblage, nous voulons que dans notre future version HPF-Halos, les mécanismes de pré-chargement et de post-stockage soient basés sur Halos et qu'ils soit implantés dans une zone bien précise et limitée du code. Pour cela, dans notre nouvelle version Fortran 90, nous créons dynamiquement un tableau *icol* pouvant contenir $\sum_{i=1}^{n}(n_1^i + n_2^i) + n_1^F$ indices,

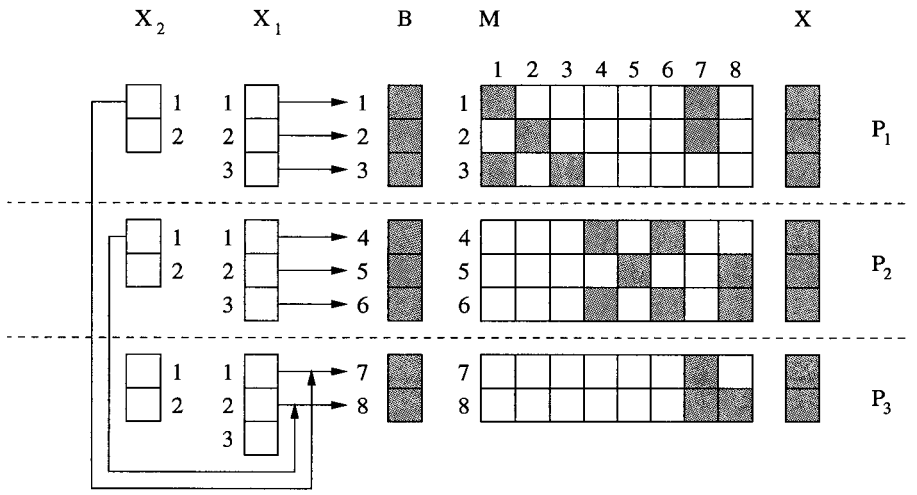

Fig. 2.17 – Post-stockage

n désignant le nombre de sous-domaines, n_1^i et n_2^i désignant respectivement le nombre d'inconnues internes et le nombre d'inconnues des interfaces du ième sous-domaine et n_1^F désignant le nombre d'inconnues internes du sous-domaine frontière. Dans notre future version HPF-Halos, ce tableau fera l'objet d'une répartition par blocs généralisée. La taille du bloc associé au ième sous-domaine sera $n_1^i + n_2^i$ tandis que celle du bloc associé au sous-domaine frontière sera n_1^F. Dès lors, les sous-domaines et le sous-domaine frontière pourront initialiser le tableau *icol* indépendamment les uns des autres. La tranche du tableau *itmp* associée à chaque sous-domaine se décompose en deux parties. La première contient les indices relatifs à ses inconnues internes. La seconde contient les indices relatifs à ses inconnues des interfaces. Quant au sous-domaine frontière, la tranche qui lui est associée ne contient que des indices relatifs à ses inconnues internes. Ces indices sont calculés à partir de la structure de donnée globale contenant les informations sur les sous-domaines et le sous-domaine frontière et à partir des tableaux d'indices de colonnes jA associés aux blocs diagonaux B_i et C ainsi qu'aux blocs d'interface F_i. Contrairement à la première version du code, chaque indice du tableau *icol* est unique à l'intérieur d'un sous-domaine ou du sous-domaine frontière. La figure 2.18 permet d'illustrer les mécanismes de pré-chargement et de post-stockage dans le cas de l'exemple de la figure 2.13.

Dans cette figure, un vecteur x utilisé par le gradient conjugué et le tableau d'indices *icol* ont respectivement fait l'objet d'une directive de répartition par blocs et d'une directive de répartition par blocs généralisée. Via le tableau *icol*, les sous-domaines et le sous-domaine frontière peuvent prélever les entrées du vecteur x qui les intéressent et les stocker dans les sous-vecteurs x_1 et x_2 qui leur sont associés (pré-chargement), ou effectuer l'opération inverse, éventuellement en combinant les sous-vecteurs x_1 et x_2 (post-stockage).

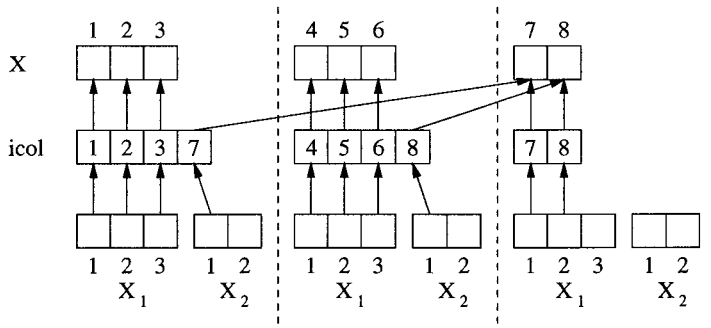

Fig. 2.18 – Mécanismes de pré-chargement et de post-stockage

2.3.9 Structure modulaire

Comme pour la version précédente du code, notre nouvelle version Fortran 90 est basée sur une structure totalement modulaire. Cette structure est présentée dans la figure 2.19.

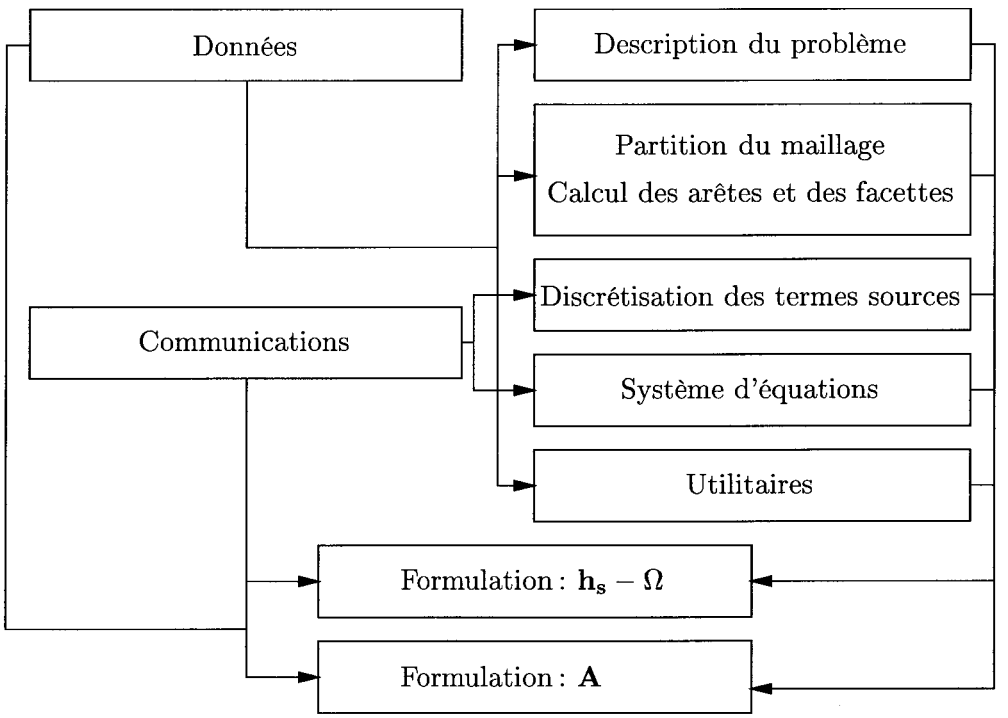

Fig. 2.19 – Structure modulaire du nouveau code

Comme précédemment, toutes les données globales de l'application sont regroupées dans un module visible depuis tous les autres. Contrairement à la première version du code ou l'ordre n'avait pas d'importance, la description du problème à traiter doit maintenant être lue avant le maillage car elle contient les informations nécessaires à son partitionnement. La partitionnement du maillage

ainsi que le calcul de ses arêtes et de ses facettes, sont regroupés au sein du même module. Les communications d'assemblage et de résolution sont également regroupées dans un module. Les communications d'assemblage sont activées par une demande de création des boites à lettres pour les sous-domaines et le sous-domaine frontière. Pour lancer une communication d'assemblage, le programmeur remplit un "bon de commande" contenant les informations à collecter, les expéditeurs et les destinataires et le transmet à la routine chargée de préparer, d'envoyer et de recevoir les messages correspondants. La figure 2.20 permet d'illustrer ce mécanisme.

```
program asm_communications_f90
use donnees, communications

  type(tBon) :: bon
  integer :: i

  call creer_boites()

  bon%circulations_hs  = .true.
  bon%flux_j           = .true.
  bon%inconnues        = .true.
  bon%vers_frontiere   = .true.
  bon%depuis_frontiere = .true.

  do i = 1, infos%nb_sous_domaines
    call preparer_messages(bon, &
                           tab_sous_domaines_1(i), &
                           tab_sous_domaines_2(i), &
                           tab_send(i))
  end do
  call asm_communications(bon)

  call detruire_boites()
end program asm_communications_f90
```

Fig. 2.20 – Utilisation des communications d'assemblage

Dans cette figure, nous supposons que le calcul des arêtes et des facettes est terminé. L'appel de la routine `creer_boites` permet de créer les tableaux *tmp* et *itmp* et d'attribuer les positions dans les tableaux *send* et *received* pour chaque sous-domaine et le sous-domaine frontière. Lorsque les boites à lettres sont créées, les communications d'assemblage sont activées. Le programmeur peut alors remplir un "bon de commande" en indiquant les informations qu'il désire voir collecter ainsi que les expéditeurs et les destinataires. Dans notre exemple, la communication est bi-directionnelle. Les sous-domaines et le sous-domaine frontière peuvent alors préparer les messages correspondants et les stocker dans les tableaux *send* qui leur sont associés, indépendamment les uns des autres. Ces derniers se trouvent dans le tableau `tab_send`. Cette préparation est l'objet de la routine `preparer_messages` qui deviendra une routine `pure`

dans notre future version HPF-Halos du code. Notons que la variable `infos` utilisée dans l'exemple est la structure de données globale permettant de centraliser les informations sur les sous-domaines et le sous-domaine frontière dont nous parlions précédemment. Notons également que pour réduire les volumes de données échangées entre les processeurs pour la lecture et le partitionnement du maillage dans la future version HPF-Halos, nous utilisons deux tableaux alloués dynamiquement pour représenter les sous-domaines et le sous-domaine frontière. Le premier tableau `tab_sous_domaines_1` contient les listes de nœuds et d'éléments. Le second tableau `tab_sous_domaines_2` contient les blocs diagonaux et les blocs d'interface. L'appel de la routine `detruire_boites` permet de supprimer les tableaux *tmp* et *itmp*.

Les communications de résolution sont activées par une demande de création des vecteurs du gradient conjugué. Le pré-chargement des sous-vecteurs x_1 et x_2 associés aux sous-domaines et au sous-domaine frontière, est effectué en passant le vecteur du gradient conjugué concerné à la routine chargée d'effectuer les prélèvements. Une routine de post-stockage est associée au produit matrice-vecteur, aux étapes de préconditionnement avec le préconditionneur par blocs de Jacobi et à celles du préconditionneur LU par blocs. Cette routine prend comme paramètre le vecteur du gradient conjugué concerné. La figure 2.21 permet d'illustrer ce mécanisme.

```
program sol_communications_f90
use donnees, communications, systeme

  integer :: i

  call creer_vecteurs()

  call pre_chargement(X)
  do i = 1, infos%nb_sous_domaines
    call matrice_vecteur(tab_sous_domaines2(i), &
                         tab_sous_vecteurs(i))
  end do
  call post_stockage_matrice_vecteur(B)

  call detruire_vecteurs()
end program sol_communications_f90
```

Fig. 2.21 – Utilisation des communications de résolution

Dans cette figure, l'appel de la routine `creer_vecteurs` permet d'allouer dynamiquement les vecteurs utilisés par le gradient conjugué ainsi que le tableau *icol*. Lorsque ces tableaux sont créés, les communications de résolution sont activées. L'appel de la routine `pre_chargement` permet aux sous-domaines et au sous-domaine frontière de prélever les entrées du vecteur x qui les concernent et de les stocker dans les sous-vecteurs x_1 et x_2 qui leur sont associés. Ces sous-vecteurs se trouvent dans le tableau `tab_sous_vecteurs`. Dans notre exemple, l'opération effectuée est un produit matrice-vecteur $b = Mx$. Dans notre future version HPF-Halos du code, la routine `matrice_vecteur` deviendra une rou-

tine `pure`. A l'issue de cette opération, les sous-vecteurs x_2 des sous-domaines et le sous-vecteur x_1 du sous-domaine frontière doivent être combinés pour obtenir le vecteur b final du gradient conjugué. C'est l'objet de la routine `post_stockage_matrice_vecteur`. L'appel de la routine `detruire_vecteurs` permet de désallouer les vecteurs du gradient conjugué ainsi que le tableau *icol*.

Les opérations liées à la résolution du système d'équations global (numérotation des inconnues, construction des blocs, calcul des factorisations incomplètes de Crout, résolutions triangulaires et produits matrice-vecteur) sont regroupées dans un module. Comme pour la version précédente du code, un module est dédié à chaque formulation magnétostatique.

2.4 Parallélisation HPF-Halos

La nouvelle version Fortran 90 du code ayant été écrite en fonction de la manière dont nous voulions la paralléliser en HPF-Halos, sa parallélisation est beaucoup plus simple à réaliser que celle de la version précédente. Le type de parallélisme employé est ici un parallélisme au niveau des sous-domaines et du sous-domaine frontière. Ces derniers étant stockés dans un tableau (en fait dans deux tableaux complémentaires), il suffit de paralléliser la boucle de parcours de ce tableau en HPF. Halos, quant à elle, est utilisée pour assurer des communications efficaces (assemblage et résolution) entres les sous-domaines et le sous-domaine frontière. Étant confinée à l'intérieur du module chargé d'assurer les communications, elle est invisible depuis l'extérieur de ce module. La version d'ADAPTOR utilisée pour créer cette nouvelle version parallèle du code est la version 7.0. Le nombre de directives (HPF) et de lignes de code (Halos) ajoutées dans la version Fortran 90 n'excède par 120, soit environ 1% du nombre de lignes initial.

La première chose à faire est d'aligner les tableaux `tab_sous_domaines_1` et `tab_sous_domaines_2` contenant les sous-domaines et le sous-domaine frontière, les tableaux `tab_send` et `tab_received` contenant les messages à envoyer et les messages reçus, ainsi que le tableau `tab_sous_vecteurs` contenant les sous-vecteurs x_1 et x_2, puis de les répartir par blocs. Nous comprenons alors tout l'intérêt des types définis par le programmeur dans les applications HPF, ces types permettant d'éviter l'emploi de la clause `resident` pour les tableaux de tailles différentes ne pouvant être alignés les uns avec les autres. Tous ces tableaux sont ensuite alloués dynamiquement en fonction du nombre de sous-domaines donné par l'utilisateur dans sa description du problème à traiter.

La lecture et le partitionnement du maillage sont effectués par une fonction précédée de l'extension `extrinsic(hpf_serial)`. Cette extension signifie que la fonction ne doit être exécutée que par un unique processeur. Ces opérations ne concernent que le tableau `tab_sous_domaines_1` contenant les tableaux de nœuds et d'éléments des sous-domaines et du sous-domaine frontière.

Trois types de boucles peuvent être rencontrés : les boucles faisant agir tous les sous-domaines et le sous-domaine frontière, celles ne faisant agir que les sous-domaines et celles ne faisant agir que le sous-domaine frontière. Les tableaux `tab_sous_domaines_1`, `tab_sous_domaines_2`, `tab_send`, `tab_received` ainsi

que `tab_sous_vecteurs` étant alignés, la directive `independent` peut être employée seule. La figure 2.22 présente les versions HPF de ces trois types de boucles.

```
      program boucles_hpf
      use donnees, ...

        integer :: i

        ! Sous-domaines et sous-domaine frontiere.
!hpf$ independent
        do i = 1, infos%nb_sous_domaines
          call fonction_pure(tab_sub_domaines_1(i), &
                             tab_sub_domaines_2(i), &
                             tab_send(i), &
                             tab_received(i))
        end do

        ! Sous-domaines.
!hpf$ independent
        do i = 1, infos%nb_sous_domaines
          if (i /= infos%nb_sous_domaines) then
            call fonction_pure(tab_sub_domaines_1(i), &
                               tab_sub_domaines_2(i), &
                               tab_send(i), &
                               tab_received(i))
          end if
        end do

        ! Sous-domaine frontiere.
!hpf$ independent
        do i = 1, infos%nb_sous_domaines
           if (i == infos%nb_sous_domaines) then
             call fonction_pure(tab_sub_domaines_1(i), &
                                tab_sub_domaines_2(i), &
                                tab_send(i), &
                                tab_received(i))
          end if
        end do
      end program boucles_hpf
```

Fig. 2.22 – Trois types de boucles

Les tableaux *tmp* et *itmp* sont alignés et font l'objet d'une directive de répartition par bloc généralisée dès leur création. Après initialisation du tableau *itmp* par les sous-domaines et le sous-domaine frontière, nous obtenons la répartition de la figure 2.15. Les tableaux *tmp* et *itmp* n'étant pas alignés avec les tableaux relatifs aux sous-domaines et au sous-domaine frontière, la clause `resident` doit être employée. L'application de la fonction `halo_all_define` aux tableaux *tmp* et *itmp*, conduit au nouveau mécanisme d'échange des messages de la figure 2.23.

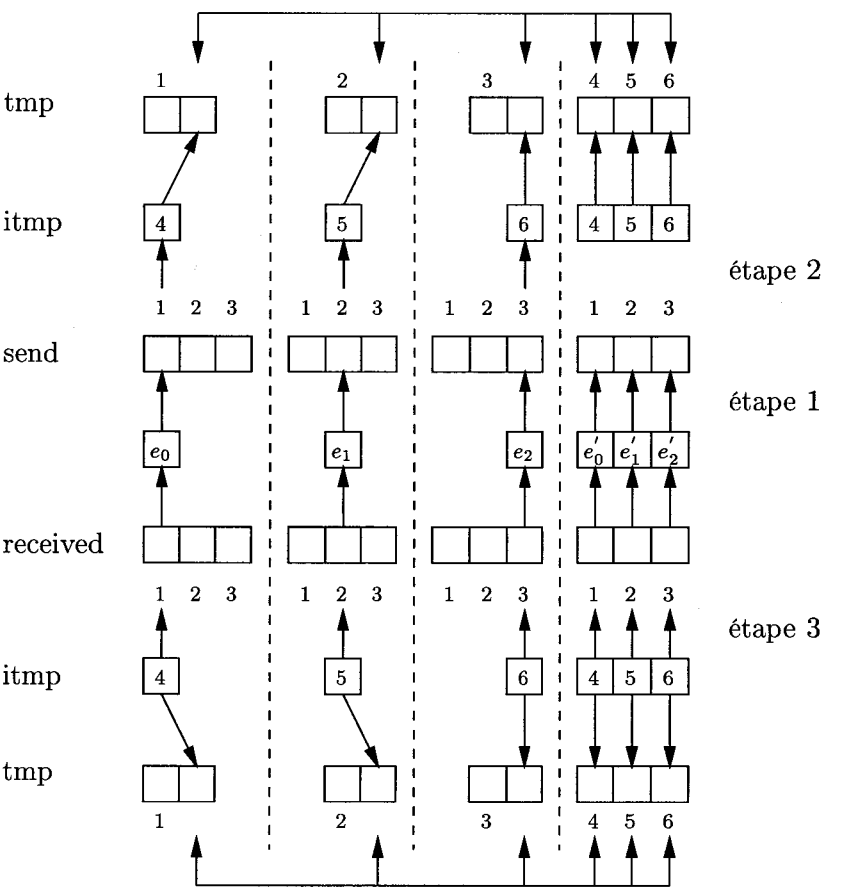

Fig. 2.23 – Mécanisme d'échange des messages basé sur Halos

Dans ce nouveau mécanisme, la fonction `halo_update` est utilisée pour faire passer les messages depuis le sous-domaine frontière vers les sous-domaines. La fonction `halo_reduce_copy` est utilisée pour faire passer les messages depuis les sous-domaines vers le sous-domaine frontière. Nous constatons alors tout l'intérêt de cette dernière fonction, fonction qui pouvait nous paraître superflue auparavant.

Comme dans la version HPF précédente du code, les vecteurs du gradient conjugué sont alignés et répartis par blocs dès leur création. Cette répartition, conjuguée à celle des tableaux relatifs aux sous-domaines et au sous-domaine frontière, conduit à la partition du système d'équations de la figure 2.16, partition qui était déjà celle de la version précédente. Le tableau *icol* fait, quant à lui, l'objet d'une directive de répartition par blocs généralisée, ainsi que nous l'avons expliqué dans la section 2.3. Après initialisation de ce dernier par les sous-domaines et le sous-domaine frontière, nous obtenons la répartition de la figure 2.18. Les vecteurs du gradient conjugué ainsi que le tableau *icol* n'étant pas alignés avec les tableaux relatifs aux sous-domaines et au sous-domaine frontière, la clause `resident` doit être employée. La fonction `halo_all_define` est appliquée à l'un des vecteurs du gradient conjugué et au tableau *icol*. La carte des communications étant la même pour tous les vecteurs du gradient conjugué, il ne reste qu'à créer les mémoires "shadow" pour les autres vecteurs, ce que nous faisons via la fonction `halo_create_"shadow"`. Nous obtenons ainsi les nouveaux mécanismes de pré-chargement et de post-stockage de la figure 2.24.

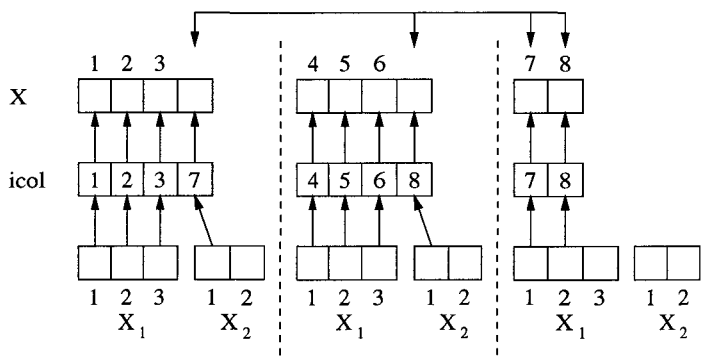

Fig. 2.24 – Mécanismes de pré-chargement et de post-stockage basés sur Halos

Le mécanisme de pré-chargement est basé sur la fonction `halo_update` qui permet de faire passer les entrées correspondantes au sous-domaine frontière vers les sous-domaines. Les mécanismes de post-stockage utilisent la fonction `halo_init` pour initialiser les mémoires "shadow" et la fonction `halo_reduce_sum` pour combiner les sous-vecteurs et faire passer les entrées correspondantes vers le sous-domaine frontière.

2.5 Résultats

La nouvelle version HPF-Halos de notre code repose sur deux hypothèses numériques. La première concerne la discrétisation des termes sources relatifs

aux inducteurs. Nous avons supposé que les arbres de facettes et d'arêtes de la version précédente, pouvaient être remplacés par des forêts d'arbres de facettes et d'arêtes, sans aucune incidence sur la solution obtenue. La seconde hypothèse concerne la matrice de préconditionnement. Nous avons supposé que le fait de résoudre le système global et non le système réduit, avec un petit nombre de sous-domaines et donc de blocs, nous permettait de nous affranchir de l'utilisation du sous-espace V_0 des méthodes de Schwarz, sous-espace que nous avons présenté en partie I. Dès lors, le préconditionneur par blocs de Jacobi qui est parfaitement adapté aux architectures parallèles, peut s'avérer suffisant. Un préconditionneur LU par blocs a cependant été prévu.

Nous avons constaté d'une part que la version parallèle précédente n'était efficace que pour un petit nombre de processeurs. Nous avons constaté d'autre part qu'elle se comportait beaucoup mieux sur une machine CC-NUMA comme l'Origin 2000 de Silicon Graphics, que sur une machine à mémoire répartie classique comme la SP2 d'IBM et ce, en raison de volumes de communications importants échangés lors de la résolution.

Le premier exemple que nous avons choisi pour tester notre nouvelle version parallèle, n'est pas un dispositif réel et sa solution peut être calculée analytiquement. Il s'agit d'un inducteur droit à section carrée, physiquement partitionné en seize sous-domaines. Tous les sous-domaines contiennent à peu près les mêmes nombres de nœuds et d'éléments et sont composés du même matériau. Le nombre d'inconnues de cet exemple, pour une formulation en potentiel vecteur magnétique, est légèrement supérieur à celui de la machine synchrone à aimants permanents utilisé pour tester la version précédente. La machine parallèle que nous avons choisie est une machine du GMD. Il s'agit d'une grappe de seize PC interconnectés par un réseau SCI. Chaque PC est un bi-processeurs Siemens Celsius 620 contenant deux Pentium Coppermine cadencés à 600 Mhz, un gigaoctets de mémoire vive et un gigaoctets de swap. Les raisons qui nous ont fait choisir cet exemple et cette machine sont multiples :

1. valider nos hypothèses numériques quant à la discrétisation des termes sources relatifs aux inducteurs et à la matrice de précondionnement ;
2. bannir les effets numériques qui sont la causes des accélérations surlinéaires obtenues avec la machine synchrone à aimants permanents sur l'Origin 2000 dans la version précédente. Il nous est ainsi possible de connaître l'impact exact de la parallélisation sur les performances de l'application, chose que nous ne pouvions faire auparavant. Ces effets numériques peuvent être minimisés en attribuant le même matériau à chaque sous-domaine ;
3. dépasser la barrière des dix processeurs avec un exemple comparable en taille à celui de la machine synchrone à aimants permanents, sur une machine qui s'apparente à la SP2 de par la répartition de sa mémoire et à l'Origin 2000 de par l'utilisation de bi-processeurs. Notons toutefois que contrairement à cette dernière, un nœud de calcul de notre grappe contient un très petit nombre de processeurs et que son mode de fonctionnement n'est absolument pas CC-NUMA.

Le second exemple choisi est un dispositif bien réel celui-là : il s'agit d'un transformateur triphasé. La version d'ADAPTOR utilisée est la version béta

8.0. Les applications générées utilisent le parallélisme de tâches et la bibliothèque de communication MPI. Le parallélisme de tâches multi-threadées n'est pas utilisé. Les deux exemples emploient une formulation en potentiel vecteur magnétique dans le cas linéaire. Le préconditionneur que nous utilisons est le préconditionneur par blocs de Jacobi. Comme pour la version précédente, tous les résultats que nous présentons ici ont été vérifiés graphiquement grâce aux fichiers de vecteurs générés par l'application et numériquement par le calcul de l'énergie électromagnétique du dispositif.

2.5.1 Inducteur droit à section carrée

La figure 2.25 présente le dispositif considéré.

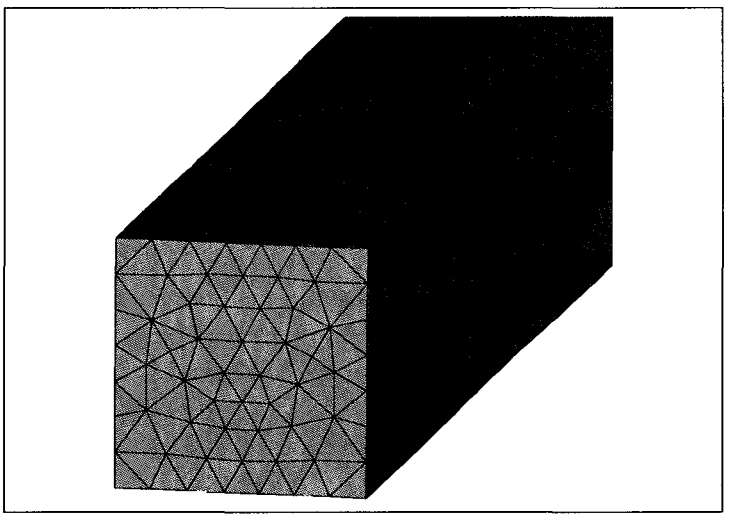

Fig. 2.25 – Inducteur droit à section carrée

Il s'agit d'un dispositif complet dont toutes les frontières sont conditionnées. La table 2.1 en présente les caractéristiques.

Tab. 2.1 – Caractéristiques du problème

nœuds	38686
arêtes	232639
facettes	371234
éléments	177280
inconnues	182750
entrées non-nulles	1448709

La table 2.2 présente les résultats obtenus pour ce dispositif. Y sont donnés, en fonction du nombre de processeurs utilisés, le temps passé à lire le maillage et à en calculer les arêtes et les facettes, le temps d'exécution de l'assemblage, le temps d'exécution de la résolution, le temps d'exécution de la formulation (discrétisation des termes sources relatifs aux inducteurs, assemblage et résolution), le nombre d'itérations du gradient conjugué et le temps passé dans chaque

itération. La table 2.3 présente les accélérations obtenues pour chaque entrée de la table 2.2.

Tab. 2.2 – Résultats sur la grappe de PC

	NP = 1	NP = 3	NP = 5	NP = 9	NP = 17
maillage	36,63 s	30,08 s	28,46 s	29,79 s	28,15 s
assemblage	18,38 s	9,16 s	4,68 s	3,45 s	1,18 s
résolution	26,40 s	16,01 s	8,47 s	6,19 s	2,95 s
formulation	53,69 s	28,31 s	14,27 s	10,49 s	4,72 s
itérations	42	47	49	48	50
temps/itération	628 ms	340 ms	171 ms	129 ms	59 ms

Tab. 2.3 – Accélérations sur la grappe de PC

	NP = 1	NP = 3	NP = 5	NP = 9	NP = 17
maillage	1,00	1,21	1,28	1,22	1,30
assemblage	1,00	2,00	3,92	5,32	15,57
résolution	1,00	1,64	3,11	4,25	8,929
formulation	1,00	1,89	3,76	5,11	11,37
itération	1,00	1,84	3,67	4,86	10,64

Nous constatons que le nombre d'itérations du gradient conjugué est relativement faible par rapport à ceux obtenus pour les dispositifs réels traités dans la version précédente du code. Ceci est dû au fait que le dispositif que nous traitons est relativement simple et que ses frontières sont parfaitement conditionnées. Le nombre d'itérations du gradient conjugué étant faible, la lecture du maillage et le calcul de ses arêtes et de ses facettes prennent alors une importance qu'ils n'ont pas dans la réalité (le temps passé pour la lecture du maillage, pour le calcul de ses arêtes et de ses facettes ainsi que pour la discrétisation des termes sources relatifs aux inducteurs, a été estimé, dans la version précédente du code, à environ 10% ou 15% du temps total d'exécution dans le cas linéaire avec une formulation en potentiel vecteur magnétique). Nous constatons également que ce nombre d'itérations n'augmente pas de manière importante avec le nombre de sous-domaines, ce qui corrobore notre seconde hypothèse. Enfin, nous remarquons que le fait d'affecter le même matériau à tous les sous-domaines a permis de minimiser au maximum les effets numériques liés à la stabilité de la matrice de préconditionnement. Il est alors possible d'estimer l'impact exact de la parallélisation sur les performances de l'application. Notons encore que pour chaque formulation magnétostatique employée, les énergies électromagnétiques calculées pour chaque partition sont parfaitement identiques.

Comme nous l'avons dit précédemment, les limites de la décomposition du domaine initial sont atteintes lorsque la charge du processeur affecté au sous-domaine frontière devient équivalente à celle des processeurs auxquels sont affectés les sous-domaines. En régime normal, le processeur affecté au sous-domaine frontière doit être sous-employé par rapport aux autres processeurs. Les accélérations obtenues pour l'assemblage et la résolution montrent que les limitations

de la version précédente ont été abolies. Ainsi, contrairement à cette dernière, l'assemblage s'adapte maintenant très bien au nombre de processeurs utilisés. Ce résultat est d'autant plus intéressant que nous n'avons pas optimisé les communications d'assemblage afin de conserver une bonne capacité de maintenance et d'évolution à notre application. L"hypothèse que nous avons formulée sur le petit nombre de ces communications était donc fondée. La forte localité des données de notre application permet de limiter aux maximum les volumes de communications échangés lors de la résolution. Le résultat obtenu est d'autant plus intéressant que les opérations autres que le produit scalaire utilisent des étapes de pré-chargement et de post-stockage qui les ralentissent d'autant et que notre grappe de PC est une machine à mémoire répartie. Dès lors, comme pour la SP2, les tâches doivent utiliser le réseau d'interconnexion pour communiquer entre elles.

Les résultats relatifs à la lecture du maillage et au calcul de ses arêtes et de ses facettes, montrent que les gains de temps obtenus dans les calculs sont perdus lors de la diffusion des listes de nœuds et d'éléments à l'ensemble des processeurs. Cette diffusion constitue donc une limitation pour la loi d'Amdahl. La solution consiste à réduire le nombre de sous-domaines ou à en augmenter la taille pour les machines à mémoire répartie, ou à tirer partie des mémoires partagées des machines à base de SMP telles que l'Origin 2000. Notons toutefois que la lecture du maillage et le calcul de ses arêtes et de ses facettes, sont des opérations qui ne sont exécutées qu'une seule fois.

2.5.2 Transformateur triphasé

La figure 2.26 présente le dispositif considéré. Il s'agit d'un transformateur triphasé. Pour des raisons de symétrie, le domaine d'étude est ramené à une moitié ce ce transformateur. Des conditions de continuité sont assurées sur le plan de symétrie. Les matériaux qui composent ce dispositif sont l'air, le fer et les enroulements primaires et secondaires.

La table 2.4 présente les caractéristiques du problème à traiter.

Tab. 2.4 – Caractéristiques du problème

nœuds	16645
arêtes	107328
facettes	177615
éléments	86931
inconnues	98383
entrées non-nulles	821388

La géométrie du dispositif fait qu'une partition physique en trois sous-domaines semble la plus naturelle. Les tables 2.5 et 2.6 présentent les résultats et les accélérations obtenus pour cet exemple.

Nous constatons que le nombre d'itérations du gradient conjugué est nettement supérieur à celui de l'exemple précédent. Ceci est dû au fait que le dispositif que nous traitons est composé de différents matériaux. Comme dans l'exemple précédent, nous remarquons que ce nombre d'itérations varie peu avec

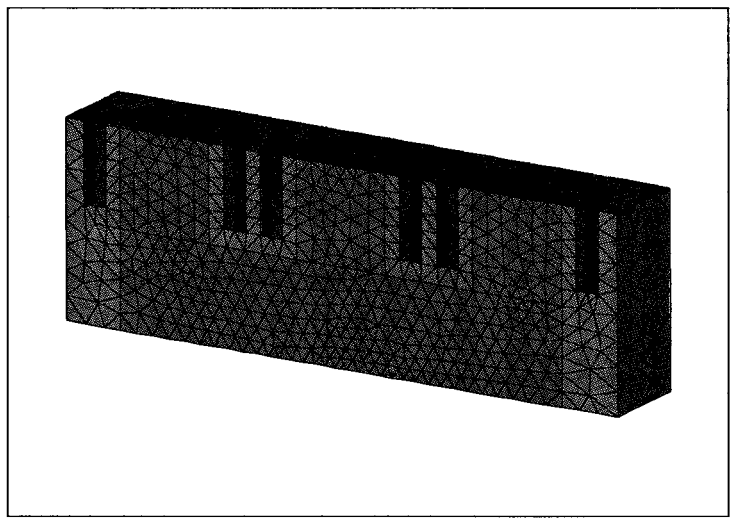

Fig. 2.26 – Transformateur triphasé

Tab. 2.5 – Résultats sur la grappe de PC

	NP = 1	NP = 4
maillage	15,21 s	13,90 s
assemblage	9,02 s	3,28 s
résolution	109,01 s	44,69 s
formulation	119,83 s	48,74 s
itérations	321	330
temps/itérations	339 ms	135 ms

le nombre de sous-domaines. Notons que ce dispositif a été également testé dans le cas non-linéaire. Là encore, les nombres d'itérations obtenus suivent la même évolution que dans le cas linéaire.

Dans cet exemple, la partition du domaine initial est réalisée de telle manière que les processeurs auxquels sont affectés les sous-domaines reçoivent à peu près la même charge de travail. Le processeur auquel est affecté le sous-domaine frontière est, quant à lui, largement sous-employé par rapport aux autres processeurs. Les accélérations que nous obtenons pour l'assemblage, pour la résolution, pour toute la formulation et pour chaque itération, montrent que l'on est proche du nombre de processeurs auxquels sont affectés les sous-domaines. Tout comme

Tab. 2.6 – Accélérations sur la grappe de PC

	NP = 1	NP = 4
maillage	1,00	1,09
assemblage	1,00	2,75
résolution	1,00	2,43
formulation	1,00	2,45
itération	1,00	2,51

dans l'exemple précédent, nous constatons que les gains obtenus dans le calcul des arêtes et des facettes du maillage sont perdus lors de la diffusion des listes de nœuds et d'éléments à l'ensemble des processeurs.

2.6 Conclusion

Les méthodes additives de Schwarz permettent de construire des matrices par blocs et des préconditionneurs associés. D'une part, la répartition d'une matrice possédant une structure par blocs sur un ensemble de processeurs, est une opération facile à réaliser. D'autre part, chaque bloc peut être vu comme une matrice indépendante, ce qui permet d'obtenir une localité des données importante. Ces deux caractéristiques expliquent l'attrait des méthodes additives de Schwarz dans le domaine du calcul parallèle scientifique. La nouvelle approche que nous proposons est basée sur ces méthodes, ce qui lui confère la localité des données qui faisait défaut à l'approche précédente. La contrepartie de cette localité est l'obligation, pour le programmeur, de modifier la structure et les algorithmes fondamentaux de l'application d'origine et donc d'écrire une nouvelle version de cette application.

Les problèmes que nous traitons étant elliptiques du second ordre, toutes les méthodes additives de Schwarz, qu'elles soient multi-niveaux ou de sous-domaines, conviennent à la résolution de nos systèmes d'équations. Pour déterminer une méthode particulière qui convienne à notre application, nous avons procédé par élimination, en fonction des différentes contraintes imposées par l'application et par les dispositifs électrotechniques traités. Dans la section 2.2 nous avons présenté ce processus d'élimination ainsi que la méthode de sous-domaines sans recouvrement retenue : la méthode du complément de Schur.

L'utilisation de la méthode du complément de Schur nous impose de paralléliser toutes les étapes de l'application d'origine, ce que nous ne faisions pas dans l'approche précédente. Cette parallélisation est assujettie à deux hypothèses numériques. La première concerne la discrétisation des termes sources et consiste à remplacer les arbres de facettes et d'arêtes de l'approche précédente, par des forêts d'arbres de facettes et d'arêtes. La seconde est relative à la résolution du système d'équations. Elle consiste à dire que les contraintes liées à l'utilisation de la méthode du complément de Schur et aux dispositifs électrotechniques traités, font que les maillages seront manuellement partitionnés par l'utilisateur et que les nombres de sous-domaines employés seront peu importants. Dès lors, la résolution du système d'équations global préconisée par Y. Saad [57] peut être effectuée sans que l'utilisation du sous-espace V_0 des méthodes de Schwarz soit nécessaire au préconditionnement, ce qui permet d'éviter les nombreuses communications entre sous-domaines générées par ce sous-espace et les problèmes liés à l'adaptation à la dimension trois des opérateurs d'interpolation sous-jacents. Dans la section 2.3, nous avons présenté une nouvelle version Fortran 90 du code construite sur ces deux hypothèses et écrite en fonction de la manière dont nous voulions la paralléliser en HPF-Halos.

Dans la section 2.4, nous avons présenté la parallélisation de la nouvelle version Fortran 90 du code. Cette version étant prévue à cet effet, sa paralléli-

sation est beaucoup plus simple à réaliser que celle de l'approche précédente. Le parallélisme employé est un parallélisme au niveau des sous-domaines et du sous-domaine frontière. Le nombre de directives (HPF) et de lignes (Halos) ajoutées dans le code n'excède pas 1% du nombre de lignes original.

Les résultats présentés dans la section 2.5 montrent que nos hypothèses numériques sont fondées. Tout d'abord, le fait de remplacer les arbres de facettes et d'arêtes de l'approche précédente par des forêts d'arbres de facettes et d'arêtes, ne modifie en rien la solution obtenue. Deuxièmement, les nombres d'itérations du gradient conjugué obtenus pour la résolution du système global sans utilisation du sous-espace V_0 dans le préconditionnement, montrent des évolutions peu importantes par rapport au nombre de sous-domaines. Ces résultats ont été obtenus sur une grappe de PC bi-processeurs interconnectés par un réseau SCI. Il s'agit donc d'une machine à mémoire répartie, bien qu'elle intègre une caractéristique SMP de par l'utilisation de bi-processeurs. Les accélérations obtenues montrent que les limitations liées à l'assemblage et à la résolution de l'approche précédente, sont maintenant abolies. Ces résultats sont intéressants pour deux raisons. La première est que nous n'avons pas optimisé les communications liées à l'assemblage, afin de préserver la facilité de maintenance et la capacité d'extension de notre application. La seconde est que nous utilisons des mécanismes de pré-chargement et de post-stockage pour les communications liées à la résolution. Ces mécanismes permettent de confiner Halos à l'intérieur d'un module "boite noire", cette dernière étant invisible depuis l'extérieur de ce module. Cependant, si de tels mécanismes répondent aux critères du génie logiciel, ils n'en ralentissent pas moins les opérations liées à la résolution. Dès lors, les résultats obtenus montrent qu'il est possible d'allier le génie logiciel à l'efficacité dans le domaine de la simulation numérique parallèle. Cette nouvelle approche impose cependant une contrainte importante à l'utilisateur : celle de devoir partitionner manuellement son maillage en se servant des plans de symétrie restants pour attribuer la même charge de travail à tous les sous-domaines. Une telle partition est facile à réaliser dans les deux exemples que nous avons utilisés. Elle l'est beaucoup moins dans l'exemple de la bobine à noyau de fer (figure 1.25) utilisé pour tester l'approche précédente. Dans cet exemple, le seul plan de symétrie du dispositif est utilisé pour n'en modéliser qu'une moitié. Dès lors, quelle que soit la partition retenue par l'utilisateur, l'équilibrage de charge sera mauvais, ce qui n'était pas le cas dans l'approche précédente puisque cet équilibrage était géré par le compilateur. Signalons tout de même que l'exemple de la bobine à noyau de fer est un cas particulier. Ainsi, dans l'exemple de la machine synchrone à aimants permanents (figure 1.26), la partition peut être facilement réalisée dans la profondeur du dispositif.

Conclusion générale et perspectives

Conclusion générale

Dans ce mémoire, nous avons présenté une approche de haut niveau permettant de paralléliser une application de simulation numérique. L'application concernée est un code de recherche basé sur les éléments finis mixtes de Whitney [9], écrit en Fortran 77 et permettant de modéliser des dispositifs électrotechniques en trois dimensions. Cette approche, basée sur l'idée de compromis entre les impératifs d'efficacité et les critères de qualité du génie logiciel, suppose l'utilisation de plusieurs outils, parfois fort différents, au sein d'une même application. Un tel compromis doit permettre à une communauté scientifique non-spécialiste d'assurer la maintenance de ce type d'application fortement évolutive. L'aspect génie logiciel est pris en charge par la nouvelle norme qu'est Fortran 90 [45] (types définis par le programmeur, modules, etc.) et par le langage à parallélisme de données standard High Performance Fortran [38]. À lui seul, HPF ne permet pas d'écrire efficacement l'intégralité de l'application. Le problème vient du fait que les codes basés sur les différences finies, les éléments finis ou les volumes finis, utilisent des matrices creuses dont les caractéristiques ne peuvent être connues qu'au moment de l'exécution. De fait, la carte des communications inter-processeurs de l'application ne peut être établie par HPF au moment de la compilation et il en résulte une gestion inefficace de ces dernières. Pour prendre en charge l'aspect efficacité, nous faisons alors appel aux bibliothèques de passage de messages. Ces dernières sont l'apanage du modèle de programmation à parallélisme de tâches, concept de plus bas niveau que le modèle à parallélisme de données, mais réputé pour son efficacité. Les bibliothèques que nous utilisons ne sont pas les bibliothèques standard PVM [35] ou MPI [36] car nous les considérons de trop bas niveau pour être abordables par des non-spécialistes. Nous préférons utiliser une bibliothèque de plus haut niveau et spécialement conçue pour être intégrée dans les applications irrégulières basées sur le modèle à parallélisme de données : Halos [3].

La première version parallèle que nous avons présentée, utilise une parallélisation très proche des méthodes de vectorisation, en ce sens que la seule opération autorisée est l'ajout, en faibles quantités, de directives HPF et de lignes de code. De fait, l'approche correspondante peut être qualifiée d'approche parallèle légère. Bien que nous sachions que les meilleures versions parallèles des applica-

tions de simulation numériques utilisent les méthodes de Schwarz, les objectifs d'une telle approche sont multiples :

- écrire une version Fortran 90 optimisée ;
- étudier les algorithmes utilisés et définir les différents types de parallélisme à mettre en œuvre ;
- valider le couple HPF-Halos ;
- repérer les changements de structures et d'algorithmes pour l'intégration future des méthodes de Schwarz.

Les résultats obtenus pour cette approche ont montré des accélérations acceptables pour un petit nombre de processeurs, ainsi que des surprises comme des accélérations surlinéaires pour un exemple relativement conséquent : une machine synchrone à aimants permanents. Ces résultats ont également fait apparaître des limitations. Tout d'abord, l'impossibilité de répartir les données pour paralléliser la phase de calcul des arêtes et des facettes du maillage ainsi que celle relative à la discrétisation des termes sources, nous a obligé à répliquer ces données sur l'ensemble des processeurs utilisés. Or, une telle manière de procéder interdit, d'une part, de traiter des exemples très importants et pénalise, d'autre part, la parallélisation de la boucle d'assemblage du système d'équations en raison de l'utilisation de la clause `reduction`. Ensuite, l'absence de localité des données fait que des volumes de communications importants sont échangés durant la phase de résolution, ce qui empêche l'application de s'exécuter sur un grand nombre de processeurs. Cependant, compte tenu du petit nombre de directives HPF et de lignes de code ajoutées (moins de 1% du nombre de lignes de la version Fortran 90), nous pouvons dire qu'avec une telle approche, l'utilisateur dispose d'une application parallèle efficace sur un petit nombre de processeurs, cette application HPF-Halos étant quasi-identique à la version Fortran 90 séquentielle, ce qui facilite grandement les opérations de maintenance. En ce sens, l'approche parallèle légère présentée est une approche de haut niveau basée sur un compromis entres les impératifs d'efficacité et les critères de qualité du génie logiciel. Cependant, ce compromis est déséquilibré au profit des critères du génie logiciel. Rappelons toutefois que nous nous attendions aux résultats obtenus, les objectifs principaux de cette approche étant de valider le couple HPF-Halos et de préparer l'intégration des méthodes de Schwarz.

La seconde version parallèle que nous avons présentée, tire les enseignements de la première en utilisant les méthodes de Schwarz. Ces dernières imposent de modifier la structure générale et les algorithmes fondamentaux de l'application et donc d'écrire une nouvelle version Fortran 90. De fait, l'approche correspondante peut être qualifiée d'approche parallèle lourde. La méthode de Schwarz que nous utilisons est la méthode du complément de Schur. Cette dernière appartient à la catégorie des méthodes de sous-domaines sans recouvrement. Contrairement à la version précédente, la phase de calcul des arêtes et des facettes du maillage est parallélisée via l'utilisation de nœuds et d'éléments "shadow". La phase de discrétisation des termes sources est, quant à elle, parallélisée via la construction de forêts d'arbres de facettes et d'arêtes. Ces forêts remplacent les arbres de facettes et d'arêtes de la version précédente. Afin de conserver une bonne facilité de maintenance, nous poussons plus loin le concept de génie logiciel en

divisant le code de l'application en zones évolutives et peu évolutives. Ces dernières sont regroupées et isolées dans des modules dont l'accès s'effectue via des interfaces imposées. Les routines de communications entre les sous-domaines appartiennent à cette catégorie. Le fait de les regrouper et de les isoler dans un module avec une interface imposée, permet au programmeur de ne jamais manipuler directement Halos. Les zones évolutives utilisent, quant à elles, des algorithmes aussi similaires que possibles à ceux de la version précédente. L'application Fortran 90 étant écrite en fonction de la manière dont nous voulons la paralléliser en HPF-Halos, cette étape est beaucoup plus facile à réaliser que dans l'approche précédente. Les résultats obtenus montrent que les limitations de la première versions sont abolies. Premièrement, le fait de paralléliser la phase de calcul des arêtes et des facettes du maillage, de même que celle relative à la discrétisation des termes sources, nous permet de ne plus répliquer les données sur l'ensemble des processeurs utilisés et donc de pouvoir traiter des exemples très importants. Deuxièment, grâce aux nœuds et aux éléments "shadow", la boucle d'assemblage du système d'équations devient une boucle totalement parallèle dans laquelle chaque sous-domaine construit la partie du système qui le concerne. Dès lors, les accélérations obtenues pour cette boucle sont proches du nombre de processeurs utilisés. Enfin, la localité des données obtenues via la méthode du complément de Schur permet de limiter au maximum les volumes de communications échangés au cours de la résolution, ce qui se traduit par des accélérations très supérieures à celles de la première version. Le regroupement des zones peu évolutives dans des modules et l'utilisation d'algorithmes quasi-similaires à ceux de la version précédente dans les zones évolutives, permet à l'utilisateur de diposer d'une application parallèle à la fois efficace et relativement simple à maintenir. Cette maintenance est cependant un peu plus complexe que pour la version précédente, le programmeur devant maintenant raisonner en termes de sous-domaines. Compte tenu du petit nombre de directives HPF et de lignes de codes ajoutées (moins de 1% du nombre de lignes de la nouvelle version Fortran 90) et du fait que les communications entre les sous–domaines n'ont volontairement pas été optimisées afin de conserver une bonne facilité de maintenance, nous pouvons dire que cette approche parallèle lourde est véritablement une approche de haut niveau permettant de concilier les impératifs d'efficacité et les critères du génie logiciel. Contrairement à l'approche précédente, ce compromis est maintenant équilibré. Il a cependant été obtenu au prix d'une refonte complète de l'application au niveau de sa structure générale et de ses algorithmes fondamentaux. Les compilateurs HPF étant libres dans le choix du modèle d'exécution, notre approche parallèle de haut niveau permet également de disposer d'une application très portable, l'adaptation à la machine cible étant réalisée par le compilateur HPF.

Perspectives

Formulations magnétodynamiques

La première suite à donner à ce travail est la ré-intégration des formulations magnétodynamiques présentées en partie I. Grâce à la décomposition de Helm-

holtz, nous savons que ces formulations peuvent également être traitées par les méthodes de Schwarz. Contrairement aux formulations magnétostatiques, la représentation du système d'équations global et la numérotation des inconnues du problèmes sont ici plus complexes en raison de l'obligation de séparer les inconnues relatives au noyau de l'opérateur différentiel concerné, des inconnues relatives à l'orthogonal de ce noyau. Cependant, certaines fonctionnalités ont déjà été intégrées à cet effet dans notre dernière version Fortran 90. La difficulté la plus importante vient de la simulation du mouvement du rotor dans une machine électrique. Il est bien sur impensable de re-mailler le dispositif à chaque pas de temps. Dès lors, ce mouvement est simulé par l'intermédiaire d'une "surface de glissement" disposée entre le rotor et le stator de la machine considérée [10, 48]. Cette surface est composée d'éléments géométriquement identiques. Le mouvement est simulé par une re-numérotation des inconnues relatives à cette suite d'éléments. Pour utiliser cette technique dans le cadre d'une application basée sur les méthodes de sous-domaines, le processus de re-numérotation ne doit pas faire "passer" un élément d'un sous-domaine à un autre. Pour éviter ce problème, la machine concernée doit être partitionnée en tranches, chaque tranche étant orthogonale à l'axe du rotor et du stator. Une telle partitition est facilement réalisable, ce que nous pouvons constater dans le cas de la machine synchrone à aimants permanents.

Machines à base de SMP

La seconde suite à donner à ce travail est d'exécuter efficacement l'application issue de l'approche parallèle lourde sur la SP3 d'IBM, machine récemment acquise par l'U.S.T.L. Il s'agit d'un calculateur à base de SMP non CC-NUMA. Nous avons défini précédemment le modèle de programmation de ce type de machine comme un modèle à parallélisme de tâches multi-threadées. La programmation multi-threadée est utilisée pour les nœuds de calcul. Lorsque ces derniers ne sont pas interconnectés via un switch CC-NUMA, les communications entre ces nœuds ne peuvent se faire que par l'intermédiaire des bibliothèques de passage de messages telles que PVM ou MPI. Lorsque ce switch est CC-NUMA, seule la programmation multi-threadée est utilisée. Notons toutefois que le modèle à parallélisme de tâches peut également être employé pour ces machines, c'est d'ailleurs sur ce modèle qu'étaient bâties les deux versions parallèles générées par ADAPTOR dont nous nous sommes servies pour obtenir nos résultats. La programmation multi-threadée des nœuds de calcul peut être mise en œuvre à partir de deux types d'outils. Le premier type d'outils est celui des pthreads : il s'agit d'outils de bas niveau mais relativement efficaces. Le second type d'outils est celui des langages de haut niveau tels qu'OpenMP [51] et plus récemment HPF [1, 52]. OpenMP est un langage véritablement conçu pour la programmation multi-threadée. Son mode de fonctionnement est basé sur le contrôle explicite du code. L'intrusion de HPF dans le domaine de la programmation multi-threadée est relativement récente. Ce n'est toutefois pas véritablement une surprise puisque les compilateurs HPF sont libres dans le choix de leur modèle d'exécution. Ainsi, la version 8.0 d'ADAPTOR, qui est actuellement en cours de finition, permet de générer du code multi-threadé à

partir de programmes OpenMP et HPF. Plusieurs combinaisons sont envisageables pour exécuter notre application sur une machine à base de SMP non CC-NUMA. Nous excluons cependant la double combinaison des bibliothèques de passage de messages et les pthreads car les applications résultants sont à notre avis bien trop complexes pour des non-spécialistes et même parfois pour des spécialistes. Parmi les options restantes, citons :

1. HPF seul avec un modèle d'exécution classique SPMD ;
2. un couplage entre MPI et OpenMP ;
3. un couplage entre MPI et HPF multi-threadé ;

Chacune de ces approches possède des avantages et des inconvénients que nous allons essayer de détailler.

1. l'utilisation du modèle SPMD permet d'exécuter notre application sans rien changer de son code. Cependant, le nombre de tâches MPI qu'il est possible d'exécuter sur un nœud de calcul est peut être inférieur au nombre de processeurs de ce nœud, ce qui conduit à une sous-utilisation de la machine. C'est notamment une des spécificités de la machine acquise par l'U.S.T.L. De plus, les communications entre les tâches se font par socket et non par l'intermédiaire de la mémoire partagée, ce qui est moins efficace ;
2. dans l'hypothèse d'un couplage entre MPI et OpenMP, le code des sous-domaines doit être multi-threadé. Ces sous-domaines sont répartis sur les nœuds de calcul et communiquent entre eux via la bibliothèque de passage de messages utilisée. Cet exemple est souvent présenté dans les introductions ou les conclusions d'articles traitant d'OpenMP. Dans notre cas, une telle approche pose difficulté. En effet, si la boucle d'assemblage d'un sous-domaine peut être facilement parallélisée en OpenMP via l'utilisation d'un verrou sur le système d'équations associé au sous-domaine, la parallélisation de la résolution est plus problèmatique. En effet, cette phase nécessite des résolutions triangulaires avec la matrice de préconditionnement associée au sous-domaine, cette dernière étant une factorisation incomplète de Crout. Or, cette matrice ne possède aucune structure particulière, c'est d'ailleurs le problème que nous avions rencontré dans notre approche parallèle légère. Pour le résoudre, nous avions utilisé un préconditionneur par blocs construit à partir d'une technique de probing. Dans l'hypothèse d'une parallélisation en OpenMP, une telle technique n'est pas utilisable car la notion de répartition des données n'existe pas dans ce langage. Des sémaphores doivent donc être employés pour assurer la cohérence des écritures, ce qui rend la résolution inefficace du fait du trop grand nombre de dépendances entre les opérations. De plus, l'absence de répartition des données du sous-domaine risque d'engendrer une mauvaise utilisation des mémoires cache. Notons qu'à l'heure actuelle, il semble se dégager un consensus sur l'intégration de directives de répartition des données dans OpenMP. L'avantage d'utiliser ce langage est lié au fait qu'il est actuellement présent sur de nombreuses machines ;
3. dans l'hypothèse d'un couplage entre MPI et HPF multi-threadé, les données des sous-domaines doivent faire l'objet d'une directive de répartition. Deux possibilités sont envisageables. La première consiste à ne pas par-

titionner les sous-domaines en sous-domaines plus petits et donc appliquer les techniques de l'approche parallèle légère aux données des sous-domaines. La seconde possibilité consiste à partitionner les sous-domaines en sous-domaines plus petits et donc appliquer les techniques de l'approche parallèle lourde aux données des sous-domaines. Cette dernière possibilité impose une nouvelle contrainte à l'utilisateur : celle d'équilibrer les charges de travail à l'intérieur du sous-domaine. Tout le problème de cette approche est de disposer d'un compilateur HPF permettant de générer du code multi-threadé efficace.

Nous constatons que la meilleure approche semble être une approche permettant d'allier les bibliothèques de passage de message à un langage de haut niveau pour la programmation multi-threadée, ce langage devant contenir des mécanismes de répartition des données. Cependant, dans le cas d'OpenMP, le langage ne dispose toujours pas de ces mécanismes. Dans le cas de HPF, les compilateurs permettant de générer du code multi-threadé ne sont pas encore vraiment efficaces.

Méthodologie de programmation

La troisième et dernière suite à donner à ce travail est son application à d'autres codes basés sur les éléments finis. Ce travail a permis de créer une application bâtie sur l'idée de compromis entre les impératifs d'efficacité de la simulation numérique et les critères de qualité du génie logiciel nécessaires à la maintenance des codes évolutifs. Les caractéristiques de cette application sont :

- un travail de développement et de mise au point réalisé à plus de 95% sur la version Fortran 90 séquentielle ;
- un nombre de directives HPF et de lignes de code ajoutées n'excédant pas 1% du nombre de lignes de la version Fortran 90 ;
- une très forte modularité et une utilisation intensive des types définis par le programmeur ainsi que des opérations vectorielles ;
- un isolement des fonctionnalités peu évolutives dans des modules dont l'accès s'effectue via des interfaces imposées. Une telle méthode permet de regrouper et d'isoler toutes les routines de communications entre les sous-domaines à l'intérieur de l'un de ces modules ;
- une utilisation de matrices et de préconditionneurs par blocs. Une telle structure permet non seulement une résolution itérative efficace sur les machines parallèles, mais également de réduire l'instabilité de la matrice de préconditionnement liée aux sauts de coefficients ;
- une bonne éfficacité obtenue avec des communications entre les sous-domaines qui n'ont volontairement pas été optimisées afin de conserver une bonne facilité de maintenance ;
- une bonne portabilité sur différentes plateformes, l'adaptation à la machine étant réalisée par le compilateur HPF.

De notre application, seule la discrétisation des termes sources et l'utilisation des éléments finis de Whitney sont réellement spécifiques. Par contre, l'utilisation des nœuds ou des éléments "shadow", le pré-chargement, le post-stockage,

le regroupement et l'isolement de routines de communication ainsi que la forte modularité, sont des techniques exportables à d'autres codes éléments finis.

Bibliographie

[1] ADAPTOR. High Performance Fortran Compilation System. WWW documentation, Institute for Algorithms and Scientific Computing (SCAI, GMD), 1999. http://www.gmd.de/SCAI/lab/adaptor.

[2] A. Albanese et G. Rubinacci. Magnetostatic Field Computations in Terms of two Components Vector Potentials. Dans *International Journal for the Numerical Methods in Engineering*, volume 29, pages 515–532, 1990.

[3] S. Benker. HPF+: High Performance Fortran for Advanced Scientific and Engineering Applications. Dans *Future Generation Computer Systems*, volume 15, pages 381–391. 1999.

[4] S. Benker. Optimizing Irregular HPF Applications Using Halos. Dans J. et al. Rolim, éditeur, *Parallel and Distributed Processing, Proceedings of IPPS/SPDP Workshops*, volume 1586 de *Lecture Notes in Computer Science*, pages 1015–1024. Springer, 1999.

[5] S. Benker. VFC: the Vienna Fortran Compiler. Dans *Scientific Programming*, volume 7, pages 67–81. 1999.

[6] D. Benzerga. *Contribution à la modélisation 3D des phénomènes électromagnétiques régis par les équations de la magnétostatique et de la magnétodynamique.* Thèse de doctorat, LGEP, Université de Paris VI, 1993.

[7] J.L. Yao Bi., L. Nicolas et A. Nicolas. H(curl) Elements on Hexahedral and Vector A.B.C.'s for Unbounded Microwave Problems. Dans *IEEE Trans. Mag.*, volume 31, pages 1538–1541, 1995.

[8] A. Bossavit. A Rational for Edge Elements in 3D Fields Computations. Dans *IEEE Trans. Mag.*, volume 24, pages 74–79, 1988.

[9] A. Bossavit. *Notions de géométrie différentielle pour l'étude des courants de Foucault et des méthodes numériques en électromagnétisme.* Eyrolles, 1991.

[10] B. Boualem. *Contribution à la modélisation 3D des systèmes électrotechniques à l'aide de formulations en potentiels : application à la machine asynchrone.* Thèse de doctorat, L2EP, Université des Sciences et Technologies de Lille, 1997.

[11] P. Boulet. *Outils pour la parallélisation automatique.* Thèse de doctorat, Laboratoire de l'Informatique du Parallélisme, E.N.S. Lyon, 1996.

[12] P. Boulet et T. Brandes. Evaluation of Automatic Parallelization Strategies for HPF Compilers. Dans *High-Performance Computing and Networking*, volume 1067 de *Lecture Notes in Computer Science*, pages 778–783, 1996.

[13] A. De La Bourdonnaye. Géométrie différentielle et éléments finis : l'exemple de l'électromagnétisme. Rapport Technique 95-42, CERMICS, 1995.

[14] A. De La Bourdonnaye. A Substructuring Method for a Harmonic Wave Propagation Problem: Analysis of the Condition Number of the Problem on the Interfaces. Rapport Technique 95-35, CERMICS, 1995.

[15] A. De La Bourdonnaye, C. Fahrat, A. Macedo, F. Magoules et F.X. Roux. A Nonoverlapping Domain Decomposition Method for the Exterior Helmholtz Problem. Rapport Technique 3271, INRIA, 1997.

[16] J.H. Bramble, J.E. Pasciak et A.H. Schatz. The Construction of Preconditioners for Elliptic Problems by Substructuring. Dans *I.Math. Comp.*, volume 47, pages 103–134, 1986.

[17] J.H. Bramble, J.E. Pasciak et J. Xu. Parallel Multilevel Precondionners. Dans *Math. Comp.*, volume 55, pages 1–22, 1990.

[18] T. Brandes. HPF Library, Language and Compiler Support for Shadow Edges in Data Parallel Irregular Computations. Dans *8th International Workshop on Compilers for Parallel Computers*, pages 21–34, Aussois, France, 2000.

[19] T. Brandes et S. Benker. Exploiting Data Locality on Scalable Shared Memory Machines With Data Parallel Programs. Dans *European Conference on Parallel Computing*, pages 647–657, Munich, Germany, 2000.

[20] T. Brandes, F. Brégier, M.C. Counhil et J. Roman. Contribution to Better Handling of Irregular Problems in HPF2. Dans *European Conference on Parallel Computing*, pages 639–649, Southampton, England, 1998.

[21] T. Brandes, F. Zimmermann, C. Borel et M. Brédif. Evaluation of High Performance Fortran for an Industrial Computational Fluid Dynamic Code. Dans *3th International Meeting on Vector and Parallel Processing*, pages 467–480, 1998.

[22] F. Brégier. Proposition de gestion de l'irrégularité avec HPF2. Dans *10ième Rencontres Francophones du Parallélisme des Architectures et des Systèmes*, pages 11–14, Strasbourg, 1998.

[23] F. Brégier. *Extensions du langage HPF pour la mise en œuvre de programmes parallèles manipulant des structures de données irrégulières.* Thèse de doctorat, Université de Bordeaux I - LaBRI, 1999.

[24] E. Cagniot, T. Brandes, J.L. Dekeyser, F. Piriou, P. Boulet et S. Clénet. High Level Parallelization of a 3D Electromagnetic Simulation Code With Irregular Communication Patterns. Dans *4th International Meeting on Vector and Parallel Processing*, pages 929–938, Porto, Portugal, 2000. Will also be published by Springer in its Lecture Notes of Computer Science series.

[25] E. Cagniot, T. Brandes, J.L. Dekeyser, F. Piriou, P. Boulet et G. Marques. Parallelization of 3D Magnetostatic Code Using High Performance Fortran. Dans *IEEE International Conference on Parallel Computing in Electrical Engineering*, pages 181–185, Trois-Rivières, Québec, Canada, 2000.

[26] L.M. Carvalho. *Méthodes du complément de Schur préconditionnées dans des environnements à mémoire distribuée.* Thèse de doctorat, Institut National Polytechnique de Toulouse, 1997.

[27] E. Cecchet. Mémoire partagée distribuée pour des grappes de calcul de grandes tailles. Dans *12ième Rencontres Francophones du Parallélisme des Architectures et des Systèmes*, pages 139–144, Besançon, 2000.

[28] J.P. Cioni. *Résolution numérique des équations de Maxwell instationnaires par une méthode de volumes finis.* Thèse de doctorat, Université de Nice Sophia Antipolis, 1995.

[29] M. Cosnard et D. Trystram. *Algorithmes et architectures parallèles.* Inter-Editions, 1993.

[30] R. Couturier. Couplage OpenMP/ MPI: une expérimentation. Dans *12ième Rencontres Francophones du Parallélisme des Architectures et des Systèmes*, pages 133–138, Besançon, 2000.

[31] P. Dular. *Modélisation du champ magnétique et des courants induits dans des systèmes tridimensionnels non linéaires.* Thèse de doctorat, Université de Liège, 1994.

[32] Y. Escaig et G. Touzot. Optimization of Direct Domain Decomposition Methods. Dans *Computer Assisted Mechanics and Engineering Sciences*, volume 3, pages 1–8, 1996.

[33] Y. Escaig, M. Vayssade et G. Touzot. Une méthode de décomposition de domaines multifrontale multiniveaux. Dans *Revue européene des éléments finis*, volume 3, pages 311–337, 1994.

[34] M.J. Flynn. *Computer Architecture, Pipelined and Parallel Processor Design.* Jones and Barlet Publishers, 1995.

[35] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek et V.S. Sunderam. *PVM: Parallel Virtual Machine - a User's Guide and Tutorial for Networked Parallel Computing.* MIT Press, 1994.

[36] W. Gropp, E. Lusk et A. Skjelum. *Using MPI: Portable Parallel Programming With the Message Passing Interface.* MIT Press, 1994.

[37] J. Hansen, P. Koch et E. Jul. A Stream Protocol Implementation for an SCI-Based Cluser of Workstation. Dans *Workshop on Cluster-Based Computing*, 1999.

[38] Version 2.0 High Performance Fortran Forum: High Performance Fortran Language Specification. Rapport Technique, Department of Computer Science, Rice University, 1997.

[39] R. Hiptmair. *Multilevel Preconditionning for Mixed Elements in Three Dimensions.* Thèse de doctorat, Mathematisches Institut, Universität Augsburg, 1996.

[40] R. Hiptmair. Multigrid Method for Maxwell's Equations. Rapport Technique 374, Mathematisches Institut, Universität Augsburg, 1997.

[41] R. Hiptmair et A. Tosseli. Overlapping Schwarz Methods for Vector Valued Elliptic Problems in Three Dimensions. Rapport Technique, Institut of Mathematical Sciences, New York University, 1997.

[42] T. Jacques, L. Nicolas et C. Vollaire. Le calcul de champs électromagnétiques sur architectures MIMD. Dans *Revue Internationale de Génie Electrique*, volume 2, pages 185–214, 1999.

[43] J. Laudon et D. Lenoski. The SGI Origin : a ccNUMA Highly Scalable Server. Dans *24th Annual International Symposium on Computer Architecture*, pages 241–251, 1997.

[44] C. Lefèbvre. *HPF-Builder : un environnement visuel de placement et distribution dédié à HPF.* Thèse de doctorat, Université des Sciences et Technologies de Lille, 1998.

[45] P. Lignelet. *Manuel complet du langage Fortran 90 et Fortran 95.* Masson, 1996.

[46] B. Lucquin et O. Pironneau. *Introduction au calcul scientifique.* Masson, 1996.

[47] G. Marques. *Contribution à l'estimation numérique en 3D : application aux problèmes d'électromagnétisme statique.* Thèse de doctorat, L2EP, Université des Sciences et Technologies de Lille, 2000.

[48] Y. Le Menach. *Contribution à la modélisation numérique des systèmes tridimensionnels électrotechniques : prise en compte des inducteurs.* Thèse de doctorat, L2EP, Université des Sciences et Technologies de Lille, 1999.

[49] Y.Le Menach, S.Clénet et F.Piriou. Determination and Utilization of the Source Field in 3D Magnetostatic Problems. Dans *IEEE Trans. Mag.*, volume 34, pages 2509–2512, 1998.

[50] F. Nataf. A Schwarz Additive Method with High Order Interface Condition and Nonoverlapping Subdomains. Rapport Technique, CMAP, Ecole Polytechnique, 1996.

[51] OpenMP: Fortran Application Program Interface. WWW documentation, 1997. http://www.openmp.org.

[52] C. Perez. *Compilation des langages à parallélisme de données : gestion de l'équilibrage de charge par un exécutif à base de processus légers.* Thèse de doctorat, Laboratoire de l'Informatique du Parallélisme, E.N.S. Lyon, 1999.

[53] S. Raïna. *Emulation of a Virtual Shared Memory Architecture.* Thèse de doctorat, University of Bristol, Department of Computer Science, 1993.

[54] Z. Ren. Influence of R.H.S. on the Convergence behaviour of Curl-Curl Equation. Dans *IEEE Trans. Mag.*, volume 32, pages 655–658, 1996.

[55] Y. Saad. Approximate Inverse Preconditioners for General Sparse Matrices. Rapport Technique 94-101, University of Minnesota, Department of Computer Science and Engineering, 1994.

[56] Y. Saad. Approximate Inverse Techniques for Block-Partitioned Matrices. Rapport Technique 95-13, University of Minnesota, Department of Computer Science and Engineering, 1995.

[57] Y. Saad. Distributed Schur Complement Techniques for General Sparse Linear Systems. Rapport Technique 97-159, University of Minnesota, Department of Computer Science and Engineering, 1997.

[58] Y. Saad. BILUTM: A Domain-Based Multi-Level Block ILUT Preconditioner for GeneralSparse Matrices. Rapport Technique 98-118, University of Minnesota, Department of Computer Science and Engineering, 1998.

[59] L. Sainsaulieu. *Calcul scientifique.* Masson, 1996.

[60] J.P. Shao. *Domain Decomposition Algorithms.* Thèse de doctorat, Department of Mathematics, University of California, Los Angeles, 1993.

[61] B.F. Smith. Domain Decomposition Algorithms for Partial Differential Equations of Linear Elasticity. Rapport Technique 517, Department of Computer Science, Courant Institute of Mathematical Sciences, 1990.

[62] B.F. Smith, P.E. Bjørstad et W. Grop. *Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations.* Cambridge University Press, 1996.

[63] P. Le Tallec, Y.H. De Roeck et M. Vidrascu. Domain Decomposition Methods for Large Linearly Elliptic Three Dimensional Problems. Rapport Technique 1182, INRIA, 1990.

[64] A. Tosseli. Overlapping Schwarz Methods for Maxwell's Equations in 3D. Rapport Technique 736, Institut of Mathematical Sciences, New York University, 1997.

[65] J. Xu. Iterative Methods by Space Decomposition and subspace Correction. Dans *SIAM Review*, volume 34, pages 581–613, 1992.

[66] J. Xu. *An Introduction to Multilevel Methods.* Oxford University Press, 1997.

[67] O.C. Zienkiewicz et R.L. Taylor. *La méthode des éléments finis. Formulations de base et problèmes linéaires.* AFNOR pour l'édition française, 1991.

Titre : Algorithmes Data-parallèles Irréguliers Appliqués à la Simulation Electromagnétique Tridimensionnelle.

Résumé : La modélisation numérique en électromagnétisme permet de réduire les coûts de développement d'un dispositif en prédisant son comportement sur ordinateur. La qualité de la prédiction dépend de la finesse du modèle employé. De nos jours, la complexité des dispositifs oblige souvent à recourir à des modèles 3D, grands consommateurs de puissance et de mémoire. Une solution consiste à employer les architectures parallèles.

Nous nous intéressons ici à la parallélisation d'une application de recherche 3D, écrite en Fortran 77 et basée sur les éléments finis de Whitney. Elle est fortement évolutive et sa maintenance est assurée par des électrotechniciens.

Nous proposons une approche de haut niveau basée sur l'idée de compromis entre les impératifs d'efficacité et les critères de qualité du génie logiciel. Le but est de créer une application parallèle efficace et facilement maintenable pour des non-spécialistes. L'aspect génie logiciel est pris en charge par la nouvelle norme qu'est Fortran 90 et par le modèle à parallélisme de données incarné par son langage standard High Performance Fortran (HPF). Ce dernier, dans le cas des applications irrégulières, ne permet pas une gestion efficace des communications. Pour rendre ces dernières efficaces, nous utilisons Halos, une bibliothèque de passage de messages de haut niveau spécialement conçue pour les applications HPF.

Deux versions parallèles ont été développées. La première utilise une parallélisation qui ne remet pas en cause la structure et les algorithmes fondamentaux du code. Les résultats obtenus sont très acceptables pour un petit nombre de processeurs, mais ne peuvent être généralisés à un plus grand nombre. La seconde est basée sur la méthode du complément de Schur qui permet de corriger les faiblesses de la première, au prix d'un investissement logiciel plus important. Les résultats obtenus révèlent une application à la fois efficace et facilement maintenable pour des non-spécialistes.

Mots-clés : génie électrique, simulation 3D, architectures parallèles, placement de données, langage HPF

Title : Irrgular Data-parallel Algorithms Applied to 3D Electromagnetic Simulation

Abstract : Numerical simulation in electromagnetism allows to reduce development costs by predicting device performance using computers. The quality of a prediction depends on the accuracy of the mathematic model that is used. Nowadays, the complexity of the devices often involves 3D models, inducing CPU power and high storage capacities. One solution is to use parallel architectures.

We are interested in the parallelization of a 3D Fortran 77 research software based on Whitney's finite elements. Such a code is in constant evolution and its mainteance is done by electrical engineers.

We propose a high level parallel approach of this application based on an agreement between efficiency and software engineering. The objective is to develop a parallel code that is both efficient and easy to maintain for electrical engineers. Software engineering is taken into account by using both Fortran 90 and the standard data-parallel programming languge High Performance Fortran (HPF). HPF cannot deal efficiently with communications in irregular applications. To bypass this problem, we use Halos, a high level library for unstructured communications in HPF applications.

Two parallel versions of the code have been developed. The fist one is based on a parallelization that doesn't change the structure nor the algorithms of the application. The results are good but for a small number of processors. order to increase this number, the second one uses the Schur complement method but the development of this new version needed more features than for the previous one. The results show that this last one provides both a good efficienty and an easy maintenance.

Keywords : electrical engineering, 3D simulation, parallel architectures, data-parallelism, High Performance Fortran